

To what extent does an Object have Action

Jiaming Ke

School of Computer Science and Informatics Cardiff University

Supervisor: Dave Marshall
Moderator: Padraig Corcoran

Acknowledgements

The great success of this project has been underpinned by the experience I gained from the CUROP project “How does a machine enjoy rugby” during summer 2016. I would like to appreciate both Professor David Marshall (School of Computer Science & Informatics, Cardiff University) and his PhD student Aled Owen (School of Computer Science & Informatics, Cardiff University) for showing me the Machine Learning world. Being interested in deep learning field, I decided to choose this project to explore more knowledge about Deep Learning and Neural Network.

I encountered numerous problems and setback while I was working on the project. Fortunately, many friends of mine offered valuable help to me and inspired me while we were discussing. I would like to thank Thomas Léost (School of Computer Science & Informatics, Cardiff University) for lending me his GTX 1070 and exploring CMake with me. Specifically, I would like to thank Zhenpeng Deng (School of Mathematics & Informatics, South China Agricultural University) and Jiaying Li (Computer Science, Aberystwyth University) for inspiring me to improve my Neural Network. I would also like to thank my friend Natasha Lau for her effort in proofreading.

Finally, I would like to express my gratitude to my supervisor, Prof. David Marshall, for his encourages and support as well as his valuable suggestion for my dissertation.

Abstract

This Final Year Report documents the work that has been completed to construct a Convolutional Neural Network and study how the object in the video affects the classification. It involves training a model for action classification, using another model to detect the objects in the videos as well as modifying the objects in the testing videos before applying them to the Convolutional Neural Network model. The primary focus of the project was to train a model that performs well in the UCF-101 dataset and conduct the experiments with it.

This project implemented training a 9-layer VGG-like Convolutional Neural Network and attempted different configuration for maximising the accuracy of action classification. To modify the region of interest of the testing video, it also combines Object Detection technology and Object Tracking algorithm to fetch the ROI from each frame of the video. Then the Convolutional Neural Network can generate the prediction for the modified videos.

The prediction result is supposed to be a high dimensional array, which cannot be directly plotted in a 2D or 3D coordinate system. To solve that problem, I used Principal Component Analysis for converting the dimension of the result and then visualised that with Python matplotlib.

Contents

1	Introduction	7
1.1	Aims of project	
1.2	Approach	
1.3	Assumptions	
2	Background	11
2.1	Current Methods	
2.1.1	Computer Vision	
2.1.2	Current CNNs	
2.1.3	Object Recognition and Tracking	
2.1.4	Data processing	
2.1.4.1	Euclidean Distance	
2.1.4.2	Principal Component Analysis	
2.2	Basic Tools Used	
2.2.1	Python	
2.2.2	OpenCV	
2.2.3	Keras	
2.2.4	TensorFlow	
2.2.5	CUDA and cuDNN	
2.2.6	H5py	
2.2.7	Scikit-learn	
3	Network Structure	28
3.1	Initial Network	
3.2	Improved Network	
3.3	Theories behind the Network	
3.3.1	The Convolution Operation	
3.3.2	Activation Function	
3.3.3	Optimizer	
3.3.3.1	SGD	
3.3.3.2	RMSprop	
3.3.4	Loss Function	
3.3.5	Batch Normalization	
3.3.6	Dropout	
3.3.7	Pooling	

3.3.8	Conv Layers	
3.3.8.1	Convolutional Layer	
3.3.8.2	Pooling Layer	
3.3.8.3	Fully-Connected Layer	
3.3.9	ConvNet Architectures	
3.3.9.1	Layer Patterns	
3.3.9.2	Layer Sizing Patterns	
4	Implementation	43
4.1	Data Pre-processing	
4.1.1	Resize Videos	
4.1.2	Write and Save the Videos	
4.1.3	Shrink the Dataset	
4.1.4	Dataset API	
4.2	Neural Network	
4.2.1	Relevant Classes in Keras	
4.2.2	Codes and Comments	
4.2.3	Configuration Attempts	
4.3	Object Detection and Object Tracking	
4.3.1	Haar Cascade	
4.3.2	BoundingBoxes	
4.3.3	Meanshift and CAMshift	
5	Results	65
5.1	Data Pre-processing	
5.2	Model Summary and Classification Performance	
5.2.1	Summary	
5.2.2	Performance	
5.3	Object Detection and Object Tracking	
5.3.1	Detection	
5.3.2	Tracking	
5.4	Video Editing	
5.5	Visualisation	
6	Conclusions	74
7	Future Work	77
7.1	Improve the Network	
7.2	Inception-v4	
7.3	Different Classifier for Detection	

8	Reflection and Learning	80
8.1	Reflection	
8.2	Learning	
	Bibliography	82

1. Introduction

In recent years, object recognition has advanced significantly with large scale benchmarks and Convolutional Neural Networks' (CNN) pushing in an excellent direction. Action recognition has made similar pushes toward large-scale benchmarks, however, due to the high cost of labelling data, it has struggled to achieve this aim.

The definition of an action is often implicit within papers and ends up being a concatenation of various appearance based on different models. To this end, we are interested in exploring to what extent actions can be described by the combinations of object detectors (simple appearance-based model). It builds on top of recent object detection work, allowing us to answer this question more conclusively and exploring various combinations and parameters for techniques for combining the output of object detectors.

1.1 Aims of Project

It is necessary to construct something that can work like a human brain. In this aspect, a neural network would be perfect for this project. After researching, there is a system called Convolutional Neural Networks (CNN) in Deep Learning domain, and it has become the method of choice for processing visual and other two-dimensional data.^[1]

Therefore, I need to train a CNN for this specific aims using the famous dataset including UCF-101 (101 classes, 13K videos) and HMDB-51 (51 classes, 6.8K videos). Ideally, a model that can distinguish all the actions of these datasets will be trained. Then, by applying testing videos to the model, the features of the videos will be extracted and the model will classify the videos according to the features.

Then, object tracking will be used to find and track the region of interests (ROI). And the testing videos will be modified in several ways, i.e. blocking the ROI using a black bounding box, flipping the ROI of that video, swapping the ROI from different videos. Finally, applying the modified videos to the CNN allows us to find out the features that affect the classification.

1.2 Approach

The approach taken for this project involves these three main stages:

1. Network Construction and Training.

The first step of Machine Learning would always be Data Processing. In UCF-101 and HMDB-51, since the size of frames and the duration of videos are various, it is essential to pre-process the videos otherwise they cannot be applied to CNN. It can be defined as appropriate if the critical information in videos is preserved while shrinking the size of frames and the number of frames.

Secondly, the pre-processed data will be used to train and test the CNN. The network uses multiple convolution layers to extract the features in the videos rather than using the STIP features provided by these two datasets. Although there are some pre-trained weights on the Internet, I insist on using Convolution3D in my network instead of Convolution2D, which means that there are no pre-trained weights on the Internet that I can use, and I have to train the model from scratch, which is obviously time-consuming.

After that, the model requires testing and optimisation. It will be tested by testing videos, which is also a part of the dataset but never be used to train the model. Apparently, having a well-trained model plays a significant role in this project. Without an outstanding performance in action recognition, the result of the following experiment will become unreliable, and the whole project might be considered

2. Object Tracking and Video Editing.

At this stage, I will block/replace the ROI of the testing videos and see how it affects the result. Since the object is moving continuously in the video rather than staying in a particular location, it is necessary to find out a way to track the object. In other words, we need to know the location of the object and the size of the bounding box in each frame.

Once the ROI of the videos is found, it is relatively easy to modify the content of the ROI. If we want to replace the ROI with a black box, assigning 0 to each pixel would be an efficient way to implement it. Flipping the ROI around or making the ROI upside down is not difficult as well. If we want to swap the ROI from different videos, the only difficulty is how to cope with the size difference of two ROI. A naive solution is to swap two

ROI in the small bounding box, and use 0 to block the rest part of the larger bounding box.

3. Result Visualisation.

For each layer of the CNN, the outcome of testing video needs to be visualised and compared with the results of the modified videos. By comparing with charts and graph, we can find out the effects caused by the modification and draw a conclusion on how an action is recognised.

1.3 Assumptions

There were four assumptions in this project while I was working on it. I have tried multiple different methods in each stage, and my assumptions do hold in most cases. However, there are always some exceptions that cannot be solved. Since the exception rate is small, therefore I believe the exception can be ignored and we can assume that those four assumptions are true during the experiment.

- **Information on the videos is preserved.**

In the data pre-processing of the first stage, all the videos will be resized, and the number of frames will be set to fixed number. Given that most action happens in the central of the frame and the most action takes a short time let's assume the compression of the video just get rid of some redundant information and preserve the important features.

- **The temporal variance of the object is essential for an action.**

There are many examples on the Internet using the Convolution 2D to implement the action recognition. Take catching and throwing, for instance, if we use Convolution 2D, which means temporal variance will not be considered in the action classification, then there is no way to distinguish catching and throwing. And that is why I insist on using Convolution 3D instead of Convolution 2D. With such an assumption, there is no doubt that the network requires 3D Convolution in each layer.

- **The model is well-trained, and the outcome of it is reliable.**

It is a significant assumption of this project. If the model is not well-trained and the performance is poor, the result we get from the poor model is biased and unreliable. However, even if the accuracy of the classification is 99%, it is still not perfect, and there is something that makes the model incorrectly predict the testing data. Therefore, in this project, we need to assume the

model is reliable. Meanwhile, it is my obligation to pursue high accuracy and increase the reliability of the model.

- **Object tracking algorithm always finds the correct object in a frame.**
In Object Detection and Object Tracking, it is not 100% that the object can be found and tracked, and the exception might influence the result of this project. To address this problem, the only way to guarantee the accuracy is to manually choose the testing videos whose ROI can be correctly found and tracked by the algorithm. In this way, not only the ROI can be precisely manipulated before applying the videos to the model, but also the results are more reliable using Object Tracking.

2. Background

In this project, two key technologies drive the whole system. They are CNN and Object Tracking. CNN is a neural network with one or more convolutional layers and fully connected layers (matching those in typical artificial neural networks). Also, it uses weights and pooling layers^[2]. Its architecture allows it to take advantage of the 2D structure of input data and shows its superior results in both image and speech applications.

Regarding definition, tracking is locating an object in successive frames of a video. Although the definition is straightforward, in Computer Vision and Machine Learning, tracking is, however, a broad term that encompasses conceptually similar but technically different ideas. From my research, I noticed that there are a few different but related approaches are studied under Object Tracking^[3]: Dense Optical Flow, Sparse Optical Flow, Kalman Filtering, Meanshift and Camshift, Single Object Tracker as well as multiple object track finding algorithms.

2.1 Current Methods

In this part, I am about to introduce some relevant Computer Vision knowledge first; then I will discuss the CNNs and the Object Tracking separately.

2.1.1 Computer Vision

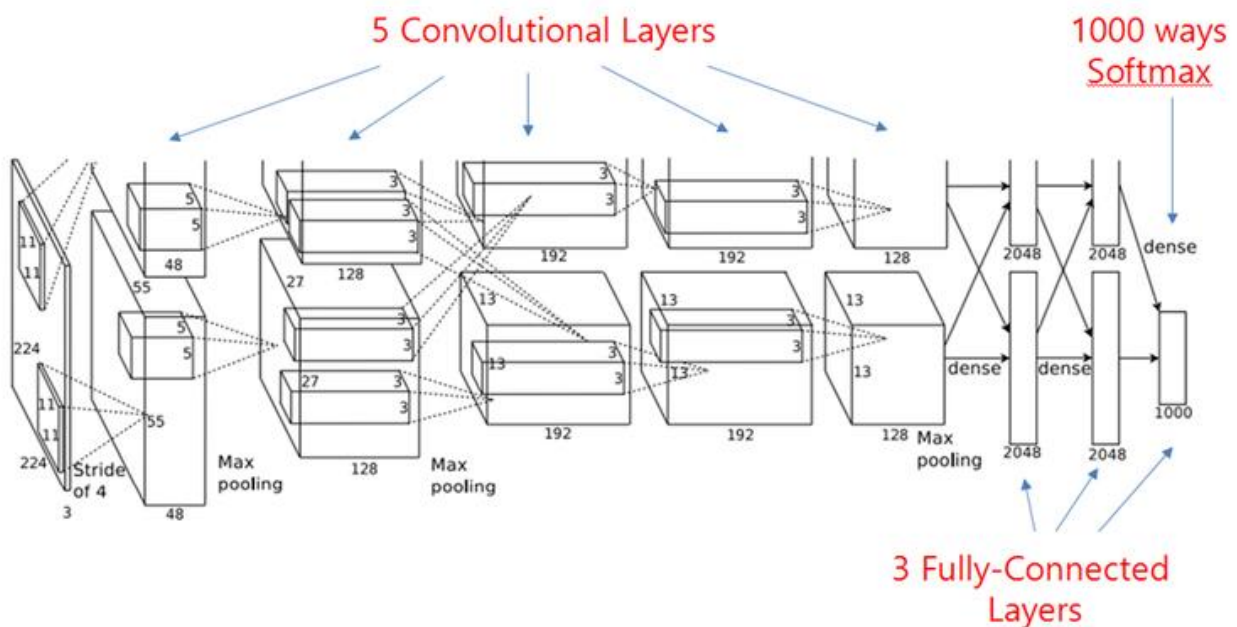
Computer vision is a very broad field encompassing a wide variety of ways of processing images and an incredible diversity of applications. Applications of computer vision range from reproducing human visual abilities, such as recognising faces, to creating entirely new categories of visual skills.^[25] Most deep learning research on computer vision focuses on a small core of AI goals aimed at replicating human abilities.

In the rest part of this section, I will discuss pre-processing. Given that sophisticated pre-processing is required by many application because original input comes in a form that is hard for representation. Some convolutional models accept variably-sized inputs and dynamically adjust the size of their pooling regions to keep the output size constant.^[30] Other convolutional models have a variable-sized output that automatically scales in size with the input, such as models that label each pixel in an image.

Pre-processing can be applied to reduce the amount of variation that the model needs to consider. Reducing the amount of variation in the data can both reduce generalisation error and lessen the size of the model required to fit the training set.

2.1.2 Relevant CNN

1. AlexNet



On one hand, it contains only 8 layers, first 5 of them are convolutional layers and the rest of them are fully connected layers. On the contrary, it seems that there are two different “streams” in AlexNet architecture because the training process is so computationally expensive and they have to split the training onto 2 GPUs.

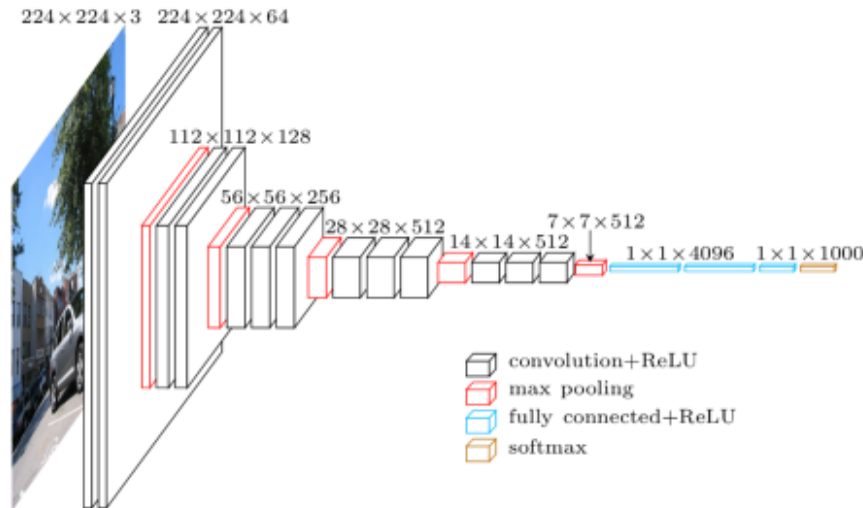
2. VGG Net

According to my research, VGGNet is designed by Visual Geometry Group, Department of Engineering Science, University of Oxford. It is a very deep convolutional network for large-scale visual recognition. As for the Visual Geometry Group, their main contribution is a rigorous evaluation of networks of increasing depth, which shows that a significant improvement on the prior art configurations can be achieved by increasing the depth to 16-19 weight layers, which is substantially deeper than what has been used in the prior art.^[4]

Regarding performance, the VGG secured the first and second places in the localisation and classification tasks respectively in ILSVRC-2014. Besides, their model did very well on PASCAL VOC and Caltech dataset.

Model	VOC-2007 (mean AP, %)	VOC-2012 (mean AP, %)	Caltech-101 (mean class recall, %)	Caltech-256 (mean class recall, %)
16-layer	89.3	89.0	91.8±1.0	85.0±0.2
19-layer	89.3	89.0	92.3±0.5	85.1±0.3
model fusion	89.7	89.3	92.7±0.5	86.2±0.3

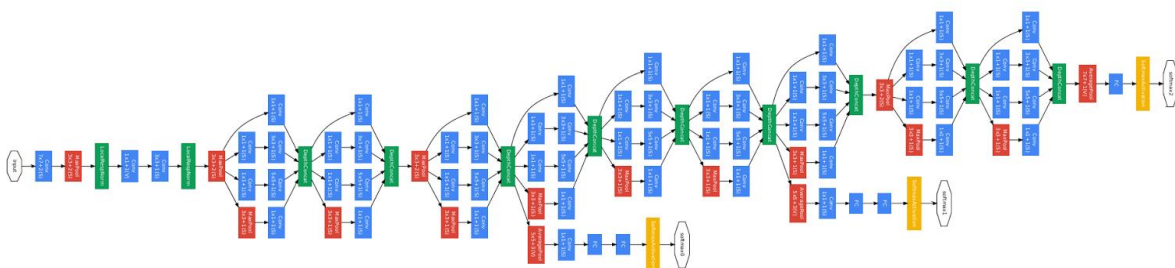
Now, let’s take VGG-16 for an example and see how its structure.



As we can see, two convolutional layers are using ReLU as activation function before the first and second max-pooling layers. For the rest max-pooling layers, there are three convolutional layers before each of them. After the last max pooling layer, three fully connected layers are used.

3. GoogLeNet

In ILSVRC 2014, there was a network called GoogLeNet which performed better than VGG did. The error of VGG was 7.32% however, that of GoogLeNet was 6.66%, which was slightly better than VGG and ranked the first in classification.



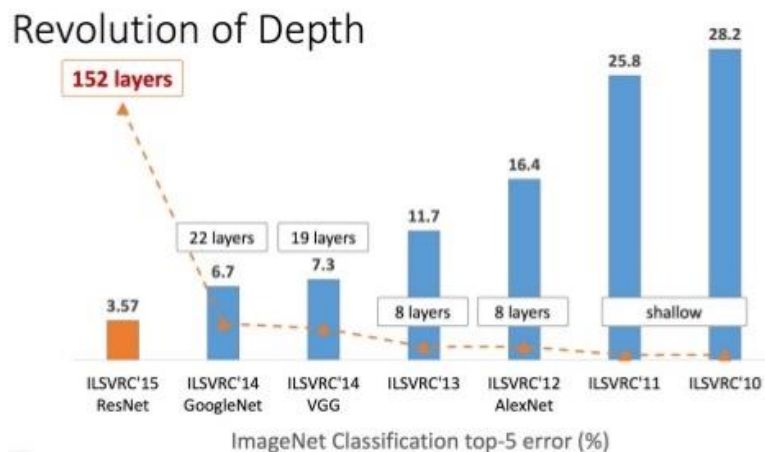
From the structure of GoogLeNet, it is not difficult to notice that the GoogLeNet is a 22-layer network. Compared with VGG, the difference is that GoogLeNet used LocalRespNorm, DepthConcat and AveragePooling while VGG did not use any of them.

Although the GoogLeNet seems complicated in structure, compared with AlexNet, it has 22 layers and almost 12x fewer parameters than Alexnet, which makes it faster and much more accurate.

4. ResNet

In ILSVRC 2015, A 152-layer residual networks (ResNet) was developed, and its error was 3.57%, which is half of that of VGG in 2014. This result won the first place on ILSVRC 2015 classification task.

E2E: Classification: ResNet



Deeper neural networks requires more time on training. Microsoft explicitly reformulates the layers as learning residual functions concerning the layer inputs, instead of learning unreferenced functions. Also, the extensive empirical evidence is shown, which proves that these residual networks are easier to optimise, and can gain accuracy from considerably increased depth. On the ImageNet dataset, the ResNet with a depth up to 152 layers was evaluated, and it achieved 3.57% error on the ImageNet test set.^[6] Compared to VGG, ResNet has lower complexity though it is 8x deeper than VGG.

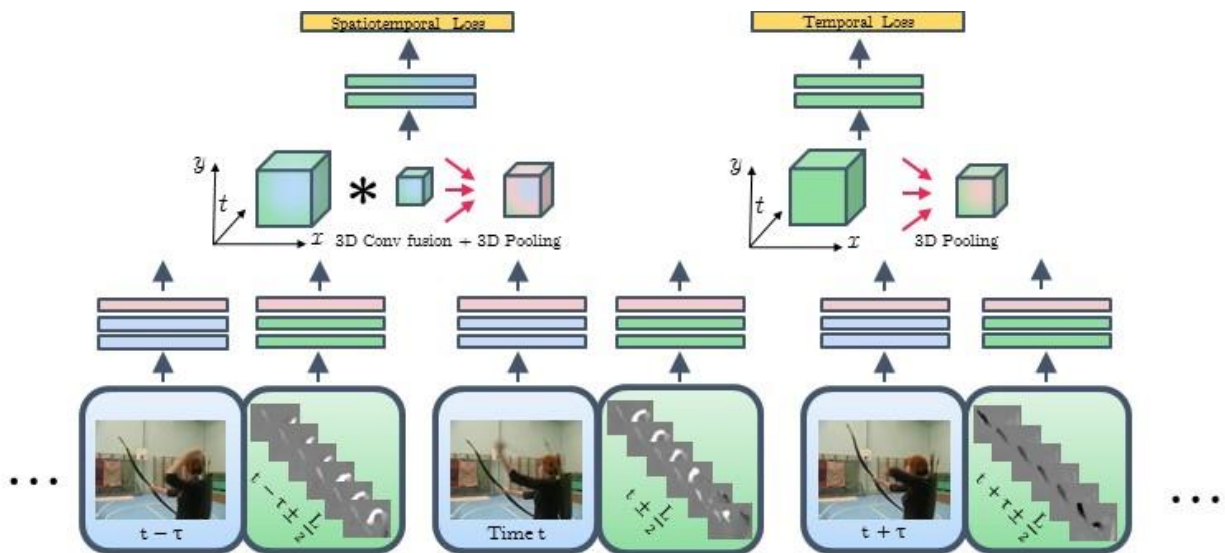
5. Two-streams VGG

In 2016, Visual Geometry Group implements human action recognition in the video using Convolutional Two-Stream Network Fusion for Video Action Recognition.[7] In that project, they decomposed video into spatial and temporal components by using still frames and optical flow information.

They used different models including deep (VGG-M), very deep (VGG-16) and extremely deep (ResNet) models to test on UCF-101 and HMDB-51.

Model	UCF-101			HMDB-51		
	spatial	temporal	both	spatial	temporal	both
<u>VGG-M</u> (ours in [1])	73.0%	83.7%	86.9%	40.5%	54.6%	58.0%
<u>VGG-16</u> (in [3] & [4])	78.4%	87.0%	91.4%	42.2%	55.0%	58.5%
<u>VGG-16</u> (ours in [2])	81.9%	88.2%	91.7%	42.6%	55.7%	58.7%
<u>ResNet-50</u> (ours in [8])	82.3%	87.0%	91.7%	48.9%	55.8%	61.2%
<u>ResNet-152</u> (ours in [8])	83.4%	87.2%	91.8%	46.7%	60.0%	63.8%

By using two-stream CNNs, the gap of classification accuracy between VGG and ResNet is significantly narrow down.



The two-stream fusion architecture above combines the two streams by a filter that can learn correspondences between highly abstract appearance features of the spatial stream (blue) and short-term motion features from the temporal stream (green). Multiple long-term input chunks are fused at the last convolutional layer. The resulting spatiotemporal features are subsequently 3D-pooled in spacetime and passed to the fully connected layers for classifying the input video.^[6]

The chart below is about the comparison of two-stream VGG and other state-of-art models.

Method	UCF-101	HMDB-51
C3D [5]	85.2%	-
Factorized ConvNet [6]	88.1%	59.1%
Two-Stream + LSTM [7] (GoogLeNet)	88.6%	-
Transformation [4] (VGG-16)	92.4%	62.0%
Two-Stream ConvNet [1] (VGG-M)	88.0%	59.4%
Two-Stream ConvNet [1] (VGG-16)	91.7%	58.5%
Two-Stream Fusion [2] (VGG-16)	92.5%	65.4%

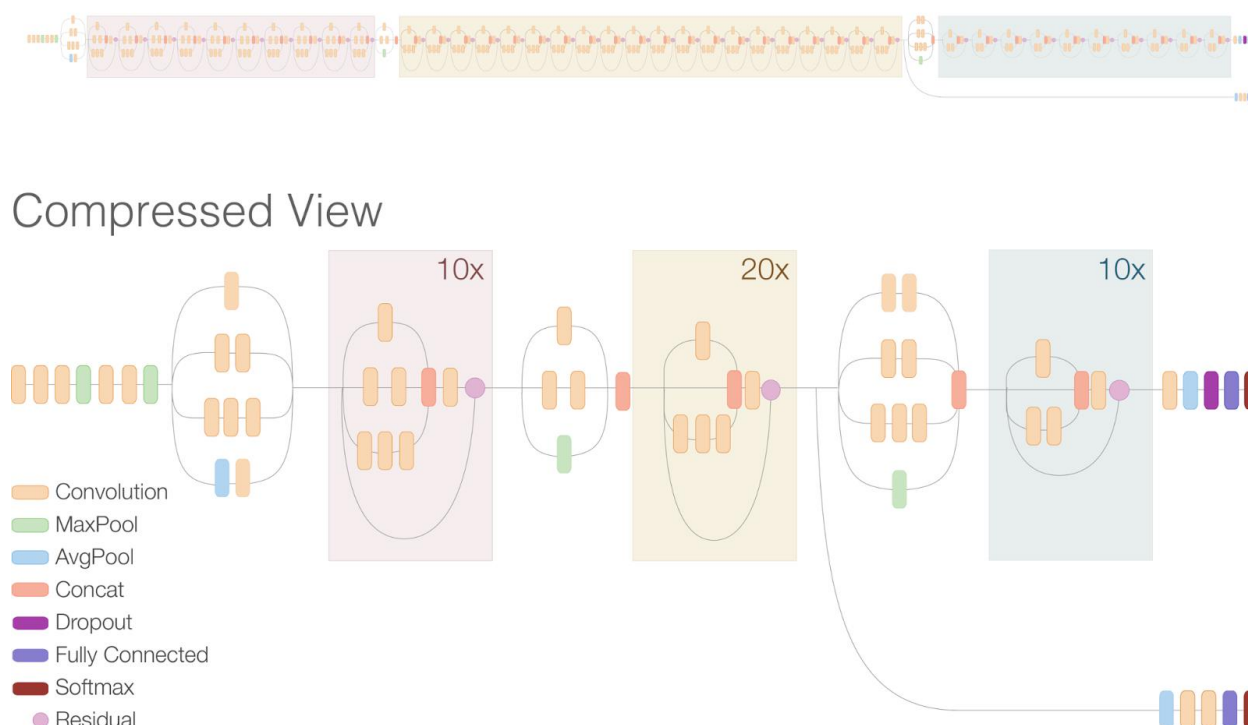
- Mean classification accuracy for state-of-the-art ConvNet approaches on UCF101 and HMDB51.

According to the chart above, the two-stream technologies does improve the performance of the model, and the Two-Stream Fusion does better than Two-Stream ConvNet in both UCF-101 and HMDB-51.

6. Inception-ResNet

On August 31, 2016, Google officially announced the release of Inception-ResNet-v2, a CNN that achieves a new state of the art regarding accuracy on the ILSVRC image classification benchmark. It is a variation of the earlier Inception V3 model which borrows some ideas from Microsoft's ResNet paper. Residual connections allow shortcuts in the model and have allowed researchers to successfully train even deeper neural networks, which have led to even better performance. It has also enabled significant simplification of the Inception blocks. ^[7]

Inception Resnet V2 Network



2.1.3 Object Recognition and Tracking

Simply speaking, tracking is locating an object in successive frames of a video. Although the definition sounds straightforward, in Computer Vision and Machine Learning, tracking is a broad term that conceptually similar but technically different. Under Object Tracking, the following ideas are different but related. ^[3]

1. **Dense Optical Flow** - help estimate the motion vector of every pixel in a video frame.
2. **Sparse Optical Flow** - track the location of a few feature points in an image.
3. **Kalman Filtering** - A popular signal processing algorithm used to predict the location of a moving object based on prior motion information.
4. **Meanshift and Camshift** - locate the maxima of a density function. They are used for tracking.
5. **Single Object Tracker** - mark the object in the first frame, using a rectangle to indicate the location of the object. Then the object is tracked in subsequent frames using the tracking algorithm.
6. **Multiple Object Track Finding Algorithms** - when we have a fast object detector, it is feasible to detect various objects in each frame and run a

tracking algorithm that identifies which rectangle in one frame corresponds to a rectangle in the next frame.

Currently, there are multiple different object tracking algorithms. Each of those has its advantages and disadvantages. When we are choosing the appropriate object tracking algorithm for our particular situation, we need to analyse the situation first then select the algorithm we need.

There are a few general principles behind tracking.^[3] In tracking, the goal is to find the object in the current frame given that we have tracked the object successfully in previous frames. Since we have tracked the object successfully until the current frame, then we must know how the object has been moving. Professionally, we are aware the parameters of the motion model, which includes the location and velocity of the object in previous frames. The new location can be predicted based on the current motion model, which is close to the actual new location.

Besides motion model, we also have other information of the object. From the previous frame, an appearance model can be built to encode the appearance of the object. It can be used to search in a small neighbourhood of the location predicted by the motion model, which is beneficial to accurately predicting the location of the object.

In a simple situation, if an object is simple and do not significantly change the appearance, a simple template can be used as a presentation model and object tracking can be implemented by looking for the template in the frames. However, the appearance of an object can change dramatically. In such a complicated situation, the appearance model is a classifier trained by an online manner. In Machine Learning, “online” refers to training the algorithms on the fly at run time. An offline classifier may need thousands of examples to train, but an online one is typically trained using a few examples at the run time.

Now let’s go through 6 different popular trackers and the current tracking algorithms behind them.

1. BOOSTING Tracker

This tracker is based on an online version of AdaBoost, the algorithm of HAAR cascade that usually used by the face detector. The classifier needs to be trained at runtime with positive and negative examples of the object. In a new frame, the classifier is run on every pixel in the

neighbourhood of the previous location, and the score of the classifier is recorded. The new location of the object is the one where the score is maximum. Then one or more positive examples are found for the classifier. As more frames come in, the classifier can be updated by the additional data in new frames.

This algorithm is a decade old, and its performance is mediocre. Besides, it does not reliably know when the tracking comes to a failure.

2. MIL Tracker

Regarding idea, the Multiple Instance Learning (MIL) tracker is similar to the BOOSTING tracker mentioned above. The difference between them is that the MIL searches a small neighbourhood around the current location to generate several potential positive examples instead of considering only the current location of the object as a positive example. It addresses the problem of tracking an object in a video given its location in the first frame and no other information.^[8]

In MIL, it is necessary to specify positive and negative bags rather than positive and negative examples. The collection of images in the positive bag are not all positive examples. In fact, only one image in the positive bag has to be a positive example. Even if the current location of the tracked object is not accurate, when samples from the neighbourhood of the current position are put in the positive bag, there is a good chance that this bag contains at least one image in which the object is nicely centred.^[3]

Compared to BOOSTING, the performance of MIL is better, and it does not drift as much as the BOOSTING tracker. Moreover, it has been shown to give promising results at real-time speeds.^[9]

3. KCF Tracker

Kernelized Correlation Filters (KCF) builds on the ideas presented above. This tracker utilises the fact that the multiple positive samples used in the MIL tracker have large overlapping regions which lead to faster and more accurate tracking.

It is better than MIL in terms of accuracy and speed. In the aspect of reporting failure, neither BOOSTING nor MIL reliably report the tracking failure while KCL does.

4. TLD Tracker

Tracking, Learning and Detection (TLD) tracker decomposes the tracking task into three components: tracking, learning and detection. The tracker follows the object from frame to frame. The detector localises all appearances that have been observed so far and corrects the tracker if necessary. The learning estimates detector's errors and updates it to avoid these mistakes in the future.^[10] Positively, this tracker appears to track an object over a large scale, motion and occlusion. If the object is hidden behind another object in the given video sequence, this tracker may be a good choice.

Therefore, this tracker works the best under occlusion over multiple frames and changing scale, but there are also numerous false positives making it a bad tracker.

5. MEDIAFLOW Tracker

This tracker tracks the object in both forward and backwards directions in time. Besides, it measures the discrepancies between these two trajectories. To reliably detect tracking failures and select reliable trajectories in a video sequence, the ForwardBackward error has to be minimised.

This tracker works best when the motion is predictable and small. Unlike other trackers, this tracker knows when the tracking has clearly failed.

6. GOTURN Tracker

It is the only one based on Convolutional Neural Network, and it also is the only one using an offline trained model, which accelerate it and make it track faster than other tracker does. It is robust to viewpoint changes, lighting change and deformations but it does not well handle the occlusion.

2.1.4 Data Processing

In this section, I would like to talk about the theories I used for processing and visualising the data. Using the technologies mentioned above, I would be able to detect and track the object in the video. Given that I can modify the video as I want, I can apply the modified video to the convolutional neural network and then store the classification results. Note that the results are always not 2-dimensional or 3-dimensional data, which means it cannot be directly visualised.

2.1.4.1 Euclidean Distance

In mathematics, the Euclidean distance is the straight-line distance between two points in Euclidean space. The associated norm is called the Euclidean norm. A generalised term for the Euclidean norm is the L^2 norm.

In Cartesian coordinates, if $\mathbf{p} = (p_1, p_2, \dots, p_n)$ and $\mathbf{q} = (q_1, q_2, \dots, q_n)$ are two points in Euclidean n -space, then the distance between them is:^[37]

$$d(\mathbf{p}, \mathbf{q}) = d(\mathbf{q}, \mathbf{p}) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots + (p_n - q_n)^2}$$

2.1.4.2 Principal Component Analysis

When I was working on the CUROP project, I used Principal Component Analysis (PCA) for dimensionality reduction. Since I would use PCA in this project, I need to introduce it and explain how to implement it below.

The main purposes of a principal component analysis are the analysis of data to identify patterns and finding patterns to reduce the dimensions of the dataset with minimal loss of information.^[33]

1. Take the whole dataset consisting of d -dimensional samples ignoring the class labels.
2. Compute the d -dimensional mean vector

3. Calculate the scatter matrix (or the covariance matrix) of the whole dataset

The scatter matrix is computed by the following equation ^[35]:

$$S = \sum_{k=1}^n (\mathbf{x}_k - \mathbf{m})(\mathbf{x}_k - \mathbf{m})^T$$

where \mathbf{m} is the mean vector:

$$\mathbf{m} = \frac{1}{n} \sum_{k=1}^n \mathbf{x}_k$$

Alternatively, we could also compute the covariance matrix using `numpy.cov()` rather than the scatter matrix. The equation for covariance matrix and scatter matrix are similar except the covariance matrix would use a scaling factor $1/N-1$.^[35] Thus their eigenspaces would be identical.

$$\Sigma_i = \begin{bmatrix} \sigma_{11}^2 & \sigma_{12}^2 & \sigma_{13}^2 \\ \sigma_{21}^2 & \sigma_{22}^2 & \sigma_{23}^2 \\ \sigma_{31}^2 & \sigma_{32}^2 & \sigma_{33}^2 \end{bmatrix}$$

4. Compute eigenvectors (e_1, e_2, \dots, e_d) and corresponding eigenvalues ($\lambda_1, \lambda_2, \dots, \lambda_d$)

The eigenvector and eigenvalue would satisfy the equation below:

$$\Sigma \mathbf{v} = \lambda \mathbf{v}$$

where

Σ = Covariance matrix

\mathbf{v} = Eigenvector

λ = Eigenvalue

5. Sort the eigenvectors by decreasing eigenvalues and choose k eigenvectors with the largest eigenvalues to form a $d \times k$ dimensional matrix W

6. **Use this $d \times k$ eigenvector matrix to transform the samples onto the new subspace**

In this step, we would use the following equation to compute the value of each point in the new subspace.

$$y = W^T \times x$$

where x is a $d \times 1$ -dimensional vector representing one sample and y is the transformed $k \times 1$ -dimensional sample in the new subspace.

2.2 Basic Tools Used

In this section, I will introduce the tools I use in this projects. For each tool, I will briefly introduce it and mention the steps for installation taking the version I installed for example. The guide on the Internet is confusing since different operating systems require different steps for installation. Although all the guide has the similar idea, it was still confusing for people especially the beginners like I was.

In the beginning, I install Python 3 and OpenCV3 in my Mac for developing. However, because of the constraint of the Linux Lab in school and the complexity of installing OpenCV, I had to use Python 2 and OpenCV2 for running the program and training the model in the server of the University. Thanks to that experience, I am not only familiar with installing OpenCV in Mac (Sierra) but also excelled at installing it in Linux, e.g. Ubuntu.

2.2.1 Python

Python is a widely used high-level interpreted programming language for general-purpose programming. I chose Python rather than the language I am familiar with, i.e. Java, Matlab, because Python has a lot of Machine Learning and Deep Learning Library that we can use. In the aspect of the experience, I can also use this project to help me improve my skills in Python programming.

Regarding version, there are two different version of Python: Python 2 and Python 3. The syntax of these two languages is different, which is the point needed to be cared about when we are programming.

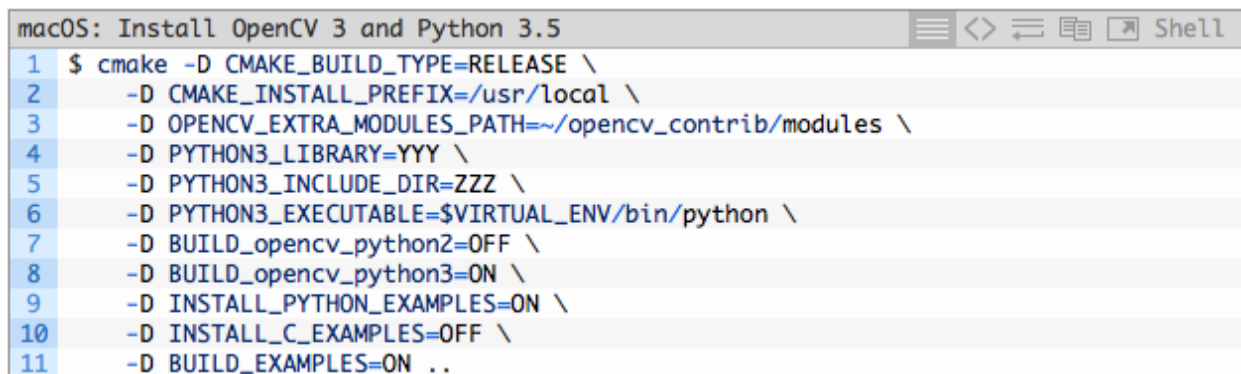
2.2.2 OpenCV

OpenCV is released under USD license, and hence it is free for both academic and commercial use. It was designed for computational efficiency and with a strong focus on real-time application. It takes the advantages of multi-core processing because of being written in optimised C/C++. Besides, it also has hardware acceleration because of OpenCL.

Similar to Python, there is multiple version of OpenCV for us to use. However, we have to obey a rule that Python 3 has to use OpenCV 3 and Python 2 has to use OpenCV 2.

There are many different guides on the Internet for installing OpenCV, but not all of them are working on your machine. It is hard to install OpenCV because the cmake is complicated to use.

For Mac OS, there is a make template for installing OpenCV 3 and Python3 constructed by A. Rosebrock^[11].

A screenshot of a macOS terminal window titled "macOS: Install OpenCV 3 and Python 3.5". The terminal shows a series of 11 lines of a cmake command. The command starts with "\$ cmake" followed by several "-D" flags. The flags are: 1. "-D CMAKE_BUILD_TYPE=RELEASE \\", 2. "-D CMAKE_INSTALL_PREFIX=/usr/local \\", 3. "-D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules \\", 4. "-D PYTHON3_LIBRARY=YYY \\", 5. "-D PYTHON3_INCLUDE_DIR=ZZZ \\", 6. "-D PYTHON3_EXECUTABLE=\$VIRTUAL_ENV/bin/python \\", 7. "-D BUILD_opencv_python2=OFF \\", 8. "-D BUILD_opencv_python3=ON \\", 9. "-D INSTALL_PYTHON_EXAMPLES=ON \\", 10. "-D INSTALL_C_EXAMPLES=OFF \\", 11. "-D BUILD_EXAMPLES=ON ..". The terminal window has a standard macOS interface with a title bar and icons for window control and a shell icon.

In the figure above, PYTHON3_LIBRARY=YYY means that we need to replace YYY with the path to your libpython3.5.dylib file, and PYTHON3_INCLUDE_DIR=ZZZ means that we need to replace ZZZ with the path to your Python.h headers. Although I followed the instruction^[11] and successfully installed OpenCV, I could import the OpenCV library and use it for reading the videos. However, I cannot save any video file, which is also an inevitable part of my project.

Since writing video is a crucial part of my project, I have no idea but to remove the OpenCV on my machine and install it again. I tried another guide^[12] for my second installation. Although the idea was the same, something was different in cmake.

Also, to enable the writing video function, build OpenCV with ffmpeg support is important. The instructions for building ffmpeg support is well described by J. Boon^[13].

When it comes to the installation of Ubuntu 16.04, the guide^[14] written by A. Rosebrock is a good one to follow. Given that I did the data pre-processing in my laptop, which includes writing and saving video, there is no need for me to set up the ffmpeg support in OpenCV again.

2.2.3 Keras

Keras is a high-level neural network API, written in Python and capable of running on top of either TensorFlow or Theano.^[15] By default, Keras will use TensorFlow as its tensor manipulation library. If TensorFlow is being used, the GPU will be automatically used for computation, which will accelerate the speed of computing.

2.2.4 TensorFlow

TensorFlow is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensor) communication between them. Originally, it was developed by Google Brain Team within Google's Machine Intelligence research organisation for conducting machine learning and deep neural networks research, but it is general enough to be applicable in a wide variety of other domains as well.^[16]

2.2.5 CUDA and cuDNN

CUDA is a parallel computing platform and application programming interface model created by NVIDIA. It enables dramatic increases in computing performance by harnessing the power of the graphics processing unit (GPU).^[17]

The NVIDIA CUDA Deep Neural Network Library (cuDNN) is a GPU-accelerated library of primitives for deep neural networks. It provides highly tuned implementations for standard routines such as forward and backwards convolution, pooling, normalisation and activation layers.^[18] It allows high-performance GPU acceleration for Deep Learning use and focuses on training neural networks rather than spending time on low-level GPU performance tuning. Therefore, in Keras

Documentation^[15], cuDNN is recommended to be installed for acceleration if the CNNs are used.

2.2.6 H5py

The h5py package is a Pythonic interface to the HDF5 binary data format.^[18] It lets you store huge amounts of numerical data, and easily manipulate that data from NumPy. Besides, It uses straightforward Numpy and Python metaphors, like dictionary and NumPy array syntax.

In Keras, it can be used to save Keras model information including the architecture of the model, the weights of the model, the training configuration (loss, optimizer) and the state if the optimizer.

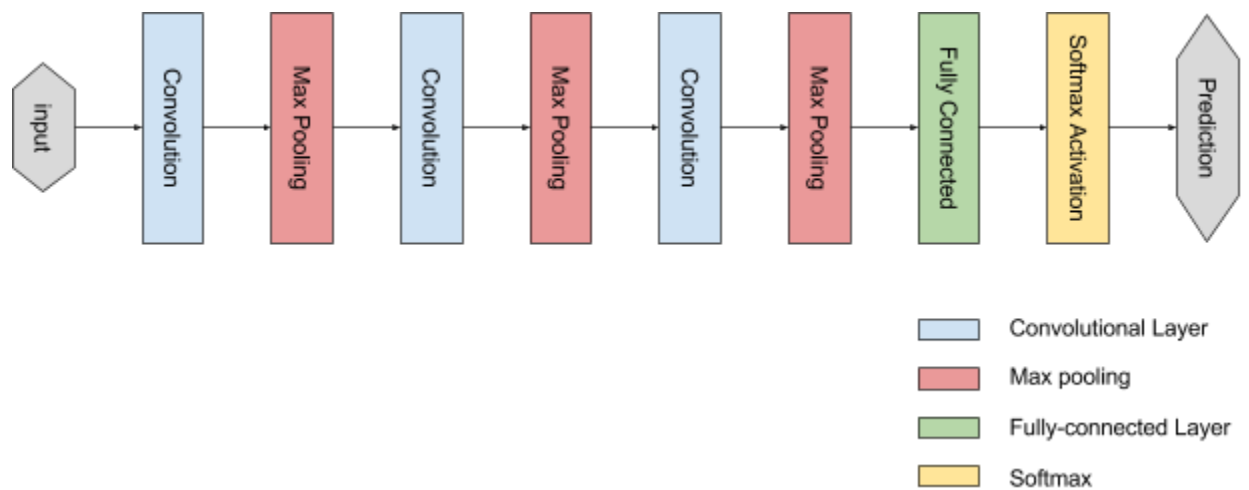
2.2.7 Scikit-learn

Scikit-learn is a free software machine learning library for the Python programming language,^[32] it implements a range of machine learning, pre-processing, cross-validation and visualisation algorithms.

3. Network Structure

In this stage, I decided to choose one of a CNN as a template and train my model for this project. After discussing with my supervisor about the network, we decided to construct a four-layer CNN for this project. However, the accuracy of the model was too poor to be used in the experiment. I tried to improve the performance by changing the configuration of the network, but the performance did not become better as I expected. Eventually, I decided to change the structure of the network and the accuracy increased significantly.

3.1 Initial Network



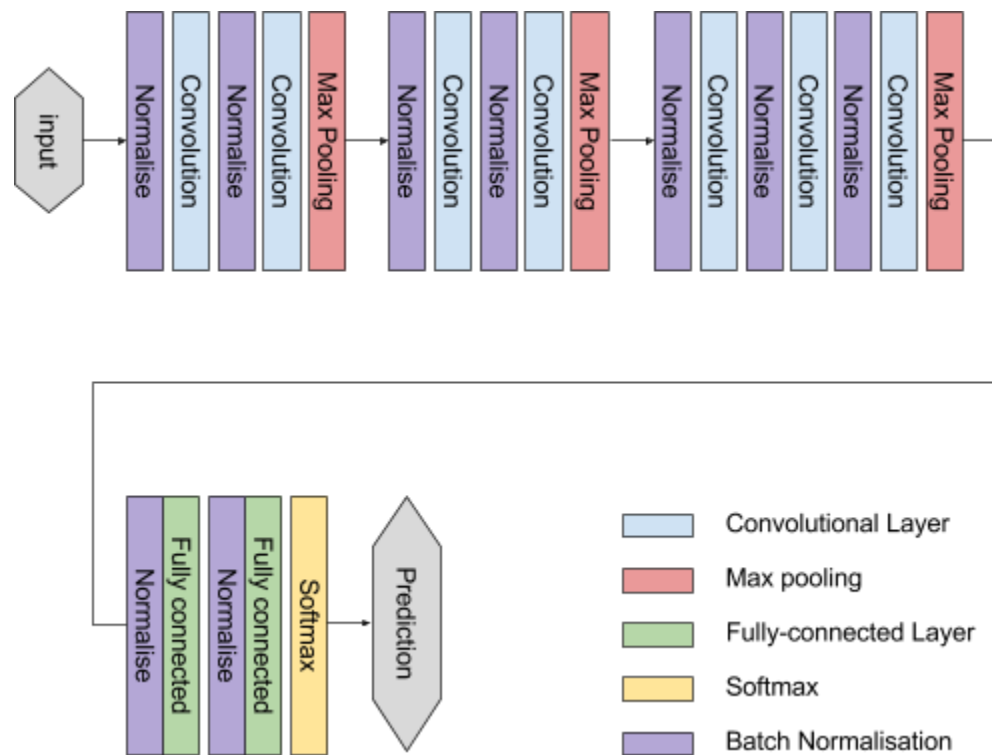
Initially, I decided a 4 layers network starting with 3 convolutional layers and ending with a fully connected layer. In the discussion with my supervisor, we thought we could be able to extract all the lines with the first convolutional layer and find the corners in the videos by applying the lines to the second convolutional layers. Then we could access to the features of the videos by applying the corners from previous layers to the third convolutional layers. Finally, with the help of the fully connected layers, the machine might be able to classify the type of action according to the combination of the features.

From what I have learned, the design seemed to be reasonable. So I spent 60 hours on training the model but the accuracy of the model could only reach 5%, which is not enough for the following experiment.

I tried different configuration of the network, hoping it can arrive at a better accuracy. After many attempts at improving the network, I eventually decided to construct a deeper network.

3.2 Improved Network

In the improved network, I amend the network with the idea I have learned from VGG net. After the reconstruction, the improved network not only become deeper than the first one but also combine both ideas from VGG net and my first CNN.



Compared to the first network, it is similar because it has four groups of layers. Compared with the VGG net, it is similar since the first three groups of layers are the convolution groups and the last group is the fully-connected group. Besides, I also added Batch Normalisation to the network for accelerating the deep network training^[19] just like GoogLeNet.

3.3 Theories behind the Network

When I was studying Deep Learning, I was recommended to read *Deep Learning*^[25] written by Ian Goodfellow. After I had started to read it, I realised that it is a resource intended to help students and practitioners enter the field of machine learning in general and deep learning in particular. Then I regarded it as my

textbook of Deep Learning. In this section, I would like to discuss the theories behind the Convolution Network based on my understanding.

3.3.1 The Convolution Operation

Convolution is an operation on two functions of a real-valued argument. It is typically denoted with an asterisk:

$$s(t) = (x * w)(t)$$

In general, convolution is defined for any functions for which the above integral is defined, and may be used for other purpose besides taking weighted averages. In convolutional network terminology, the first argument (the function x in the example above) to the convolution is often referred to as the **input** and the second argument (the function w) as the **kernel**. The output is sometimes referred to as the **feature map**.

In machine learning applications, the input is usually a multidimensional array of data and the kernel is usually a multidimensional array of parameters that are adapted by the learning algorithm. These multidimensional arrays are referred to as tensors.

We often use convolutions over more than one axis at a time, for example, if we use a two-dimensional image I as our input, we probably also want to use a two-dimensional kernel K :^[36]

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n) K(i - m, j - n).$$

Convolution is commutative, so it can also be written as:^[36]

$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i - m, j - n) K(m, n).$$

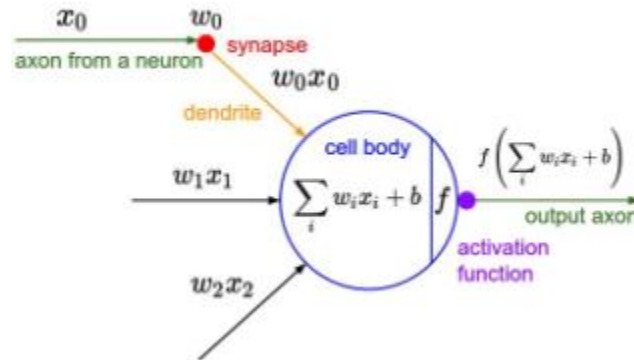
The latter formula is more straightforward to implement in a machine learning library because there is less variation in the range of valid values if m and n .

In many neural network library, there is also a related function call cross-correlation, which is the same as convolution but without flipping the kernel:^[36]

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i + m, j + n) K(m, n).$$

3.3.2 Activation Function

Before introducing activation function, I would like to start with talking about a neurone in neural networks. The graph shown below is a typical mathematical model for neurones. Each neurone receives input signals from its **dendrites** and produces output signals along its **axon**. The axon eventually branches out and connects via synapses to dendrites of other neurones.



In the mathematical model, the signals from a neurone will travel along the axons and interact with the dendrites of the other neurone based on the synaptic strength at that synapse. Based on this rate code interpretation, we model the firing rate of the neurone with an **activation function** f , which represents the frequency of the spikes along the axon.^[38] As for the choice of activation function, it is common to use the sigmoid function, since it takes a real-valued input and squashes it to range between 0 and 1.

I would start from sigmoid function and then introduce the activation function I used in my convolutional neural network.

1. Sigmoid

It is a bounded differentiable real function that defined for all real input values and has a non-negative derivative at each point.^[39] It often refers to the logistic function defined by the formula^[40]:

$$S(x) = \frac{1}{1 + e^{-x}}.$$

The sigmoid function has frequently been used since it has an excellent interpretation as the firing rate of a neurone. However, it is rarely used now since it has two significant drawbacks:

- Sigmoids saturate and kill gradients. The gradient is almost zero when the neurone's activation saturates either tail of 0 or 1. During backpropagation, the local gradient will be multiplied to the gradient of this gate's output for the whole objective, if the local gradient is small, the output will be almost 0 as well, which means almost no signal will flow through the neurone.
- The sigmoid output is not zero-centred. If the data coming into a neurone is always positive, then the gradient on the weights w will become either all positive or all negative during backpropagation, which can introduce undesirable zig-zagging dynamics in the gradient updates for weights.

2. Rectifier

A rectified linear unit (ReLU)^[41] is a unit employing the rectifier which is, as of 2015, the most famous activation function for deep neural networks.^[42] In the context of the artificial neural network, the rectifier is an activation function defined as:

$$f(x) = \max(0, x)$$

Though it is popular to use ReLU in computer vision^[43] and speech recognition^{[44] [45]} using deep neural networks, there are several pros and cons to using ReLUs:

- It was found that it can significantly accelerate (e.g. a factor of 6 in Krizhevsky's paper^[5]) the convergence of stochastic gradient descent compared to the sigmoid/tanh functions.
- It can be implemented by simply thresholding a matrix of activation at zero, while tanh/sigmoid neurones involve expensive operations.
- The ReLU units can irreversibly die during training since they can be knocked off the data manifold. In other words, if the learning rate is set too high, a lot of neurones would never activate across the entire training dataset.

3. Softmax

In mathematics, it is a logistic function that squashes a K -dimensional vector of arbitrary real values to a K -dimensional vector(\mathbf{z}) of real values in the range (0,1) that add up to 1. The function is given by:

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad \text{for } j = 1, \dots, K.$$

It is used in various multiclass classification methods^[46] including artificial neural networks. Given a sample vector \mathbf{x} and a weighting vector \mathbf{w} , the predicted probability of the j^{th} class is:

$$P(y = j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^T \mathbf{w}_j}}{\sum_{k=1}^K e^{\mathbf{x}^T \mathbf{w}_k}}$$

3.3.3 Optimizer

In this section, I would like to discuss the optimisation problems in neural networks. A vital difference between optimisation in general and optimisation we use in training algorithms is that training algorithms do not usually halt in a local minimum.

Most algorithms used for deep learning were traditionally called minibatch or minibatch stochastic methods, and it is common to call them stochastic methods simply. The canonical example of a stochastic method is stochastic gradient descent (SGD).

3.3.3.1 SGD

An essential parameter for the SGD algorithm is the learning rate. Given that the learning rate and the initial parameter, SGD can be updated by the algorithm below:

while the stopping criterion not met **do**

 Sample a mini-batch of m examples from training set $\{\mathbf{x}(1), \dots, \mathbf{x}(m)\}$ with corresponding targets $\mathbf{y}(i)$.

 Compute gradient estimate^[47]: $\hat{\mathbf{g}} \leftarrow +\frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), \mathbf{y}^{(i)})$

 Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\mathbf{g}}$

end while

A compromise between computing true gradient and the gradient at a single example is to compute the gradient against more than one training example at each step, which is why it is called minibatch stochastic method. Since the code use vectorization libraries rather than computing each step separately, estimating gradient can perform significantly better than true stochastic gradient descent.

3.3.3.2 RMSprop

Root Mean Square Propagation (RMSprop) is to divide the learning rate for weight by running average of the magnitude of recent gradients for that weight.^[48] It can be represented as:

$$w := w - \frac{\eta}{\sqrt{v(w, t)}} \nabla Q_i(w)$$

where η is the learning rate and Q is the gradient. In practice, it has shown excellent adaptation of learning rate, and it is capable of working with mini-batches.^[49]

3.3.4 Loss Function

In machine learning, loss functions for classification are computationally feasible loss functions representing the price paid for the inaccuracy of predictions in classification problems.^[51] We wish to find a function for inputs and outputs. However, it is possible for the same inputs to generate different outputs^[52] because of noise in the measurement or probabilistic components in the underlying process.

3.3.5 Batch Normalisation

It is a recently developed technique by Ioffe and Szegedy.^[19] Training Deep Neural Networks is complicated because the distribution of each layer's input changes during training, as the parameters of the previous layer change. This phenomenon is defined as *internal covariate shift*. By normalising layer inputs, Internal Covariate Shift can be reduced. Thus the Deep Network Training would be accelerated.

The algorithm of Batch Normalizing Transform is shown below:

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots x_m\}$;	
Parameters to be learned: γ, β	
Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$	
$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i$	// mini-batch mean
$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2$	// mini-batch variance
$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$	// normalize
$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i)$	// scale and shift

Firstly we will normalise each scalar feature independently by making it have the mean of zero and the variance of 1. For a layer with d -dimensional input $x = (x^{(1)} \dots x^{(d)})$, we will normalize each dimension

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

where the expectation and variance are computed over the training set. Such normalisation speeds up convergence even if the features are not decorrelated. ^[50]

Then, for each activation $x^{(k)}$, a pair of parameters $(\gamma^{(k)} \text{ and } \beta^{(k)})$ are introduced, which scale and shift the normalised value:

$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}$$

These parameters are learned along with the original model parameters and store the representation power of the network.

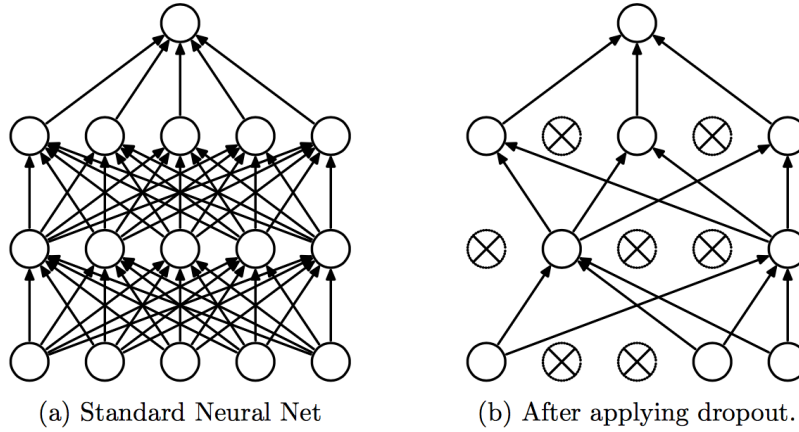
In the batch setting where each training step is based on the entire training set, we would use the whole set to normalise activations. However, this is impractical when using stochastic optimisation. Therefore, a simplification is to use the

estimates of the mean and the variance of each activation produced by each mini-batch.

In the implementation, applying this technique usually amounts to insert the BatchNorm layer immediately after fully connected layers (or convolutional layers), and before nonlinearities.

3.3.6 Dropout

Dropout is a technique to randomly drop units (along with their connections) from the neural network during training, which significantly reduces overfitting and gives major improvements over other regularisation methods.^[22]



Assume that we have a neural network with L hidden layers, let $\mathbf{z}^{(l)}$ denote the input vector of layer l and $\mathbf{y}^{(l)}$ denote the output vector. \mathbf{W} and \mathbf{b} are the weights and biases at layer l . Therefore a standard neural network can be described as:

$$\begin{aligned} z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)} \mathbf{y}^l + b_i^{(l+1)}, \\ y_i^{(l+1)} &= f(z_i^{(l+1)}), \end{aligned}$$

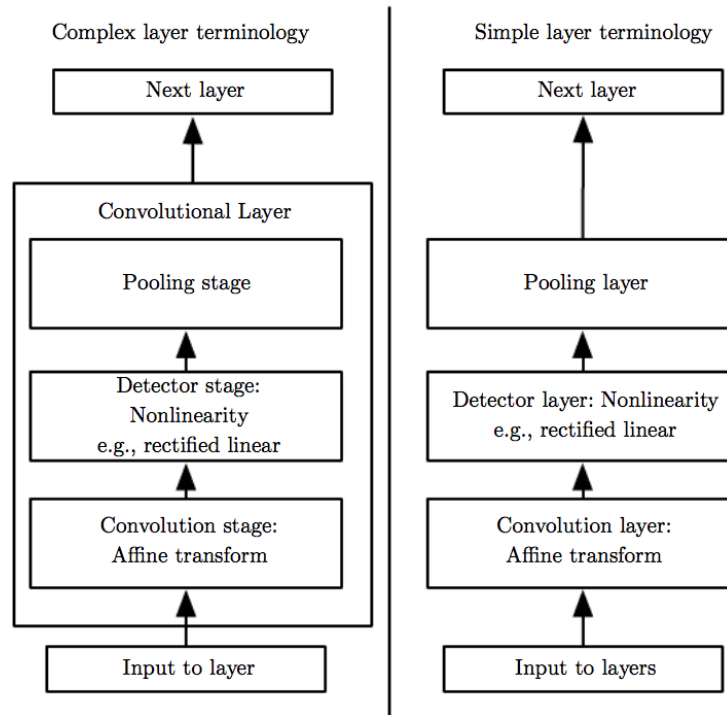
where f is any activation function.

With dropout, the feed-forward operation becomes

$$\begin{aligned} r_j^{(l)} &\sim \text{Bernoulli}(p), \\ \tilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{y}^{(l)}, \\ z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)} \tilde{\mathbf{y}}^l + b_i^{(l+1)}, \\ y_i^{(l+1)} &= f(z_i^{(l+1)}). \end{aligned}$$

3.3.7 Pooling

A typical layer of a CNN consists three stages: the layer performing convolutions produces a set of linear activations; then each linear activation is run through a nonlinear activation function, which is called detector stage; in the third stage, use a pooling function to modify the output of the layer further.



A pooling function replaces the output of the net at a certain location with a summary statistics of the nearby outputs.^[25] Take the max pooling^[34] I used in my network for example. The max pooling operation returns the maximum output within a rectangular neighborhood. Other modern pooling functions include the

average of a rectangular neighbourhood, the L^2 norm of a rectangular neighbourhood, or a weighted average based on the distance from the central pixel.

Pooling makes the representation approximately invariant to small translations of the input. Invariance to the translation means that the values of most of the pooled outputs do not change even if we translate the input by a small amount.

It is possible to use fewer pooling units than detector units by reporting summary Statistics for pooling regions spaced k pixels apart rather than 1 pixel apart because pooling summarises the responses over a whole neighbourhood. It improves the computational efficiency of the network since the next layer has roughly k times fewer inputs to process. When the number of parameters in the next layer is a function of its input size, such as when the next layer is fully connected and based on matrix multiplication, statistical efficiency can be improved, and memory requirements for storing parameters can be reduced because of that reduction.

In many cases, pooling is essential for handling inputs with various size. It is usually accomplished by varying the size of an offset between pooling regions so that the same number of summary statistics will be received.

3.3.8 Conv Layers

In this part, I will explain all the layers I used in my model, which are Convolutional Layer, Pooling Layer, Normalization Layer and Fully-connected Layer.

3.3.8.1 Convolutional Layer

The Convolutional Layer's parameters consist of a set of learnable filters. Every filter is small spatially (along width and height) but extends through the full depth of the input volume. As we slide the filter over the width and height of the input volume A 2-dimensional activation map giving the responses of that filter at every spatial position will be produced. Intuitively, the network will learn filters that activate when they see some visual features such as an edge of some orientation or a blotch of some colour on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network.

For images, each neurone will be connected to a local region of input. The spatial extent of this connectivity is a hyperparameter called the receptive field of the neurone. And the depth axis of that is always equal to the depth of the input

volume. The connections are local in space (along with width and height), but always full along with the entire depth of the input volume.

Now I would like to introduce the hyperparameters of convolutional layers which control the size of the output volume: the **depth**, **stride** and **zero-padding**.

1. The depth corresponds to the number of filters we would like to use. We refer to a set of neurones that are all viewing the same region of input as a depth column (fibre).
2. The stride equals to the number of pixels that the filters move each time. This will produce smaller output volumes spatially.
3. The zero-padding allows us to control the spatial size of the output volumes by adding zeros to the border of input.

The number of the output can be computed by the following formula:

$$f = (W - F + 2P)/S + 1$$

where f represent the number of output, W refers to the input volume size, F refers to the receptive field size, P refers to the amount of zero-padding and S means the stride. For example, for a 7x7 input and a 3x3 filter with stride 1 and pad 0, we would get a 5x5 output.

In general, when the $S = 1$, setting the zero-padding by the following function can ensure that the input volume and output volume have the same size spatially:

$$P = (F-1) / 2$$

The spatial arrangement hyperparameters have mutual constraints. If the value of f in the equation above is not an integer, it indicates that the neurones do not fit neatly and symmetrically across the input.

In Convolutional Layers, there is a Parameter sharing scheme used to control the number of parameters. Let's suppose we have a Convolutional Layer using receptive size $F = 11$, stride $S = 4$ and no zero padding $P = 0$.^[5] Besides, the Convolutional Layer has a depth $K = 96$. Assume our input data shape is [227x227x3], then the output should be [55x55x96], since $(227 - 11 + 0)/4 + 1 = 55$. There are $55 * 55 * 96 = 290,400$ neurons in the Convolutional Layer, each of them has $11 * 11 * 3 = 363$ weights and 1 bias. On total, there are $290,400 * 364 = 105,705,600$ parameters on the first layers.

To summarise, the Convolutional Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
 - Number of filters K ,
 - Their spatial extent F ,
 - The stride S ,
 - The amount of zero padding P .
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
 - $W_2 = (W_1 - F + 2P) / S + 1$
 - $H_2 = (H_1 - F + 2P) / S + 1$ (i.e. width and height are computed equally by symmetry)
 - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d -th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d -th filter over the input volume with a stride of S and then offset by d -th bias.

3.3.4.2 Pooling Layer

It is common to periodically insert a Pooling layer in-between successive Conv layers in a ConvNet architecture. Not only can it effectively reduce the number of parameters by reducing the spatial size of the representation, but it also controls overfitting. The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation.

The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2, which downsample every depth slice in the input by two along both width and height, discarding 75% of the activations. The depth dimension remains unchanged. More generally, the pooling layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
 - their spatial extent F ,
 - the stride S ,

- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
 - $W_2 = (W_1 - F) / S + 1$
 - $H_2 = (H_1 - F) / S + 1$
 - $D_2 = D_1$
- Do not introduce any parameter since it is a fix function
- Note that it is not common to use zero-padding for Pooling layers

3.3.4.3 Fully-Connected Layer

Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks. Hence their activations can be computed with a matrix multiplication followed by a bias offset.

3.3.9 ConvNet Architecture

In this section, I would like to discuss how to stack the Conv Layers together to form an entire Convolution Network.

3.3.5.1 Layer Patterns

Nowadays, the most common ConvNet architecture follows the pattern:

INPUT -> [[CONV -> RELU] * N -> POOL] * M -> [FC -> RELU] * K -> FC

Moreover, $N \geq 0$ (usually $N \leq 3$), $M \geq 0$, $K \geq 0$ (usually $K < 3$).

A stack of small filter convolutional layers has fewer parameters than one large receptive field convolutional layers. Suppose you stack three 3x3 convolutional layers, for the first layer, it has a 3x3 view of input volume; and the second layer also has a 3x3 view of the first layer, hence by extension a 5x5 view of input of volume. Thus the third layer has a 7x7 view of input volume by extension. Compared that to one single 7x7 convolution layer, a stack of small filter convolutional layers makes the features more expressive, and it also consumes less memory. Assume all the volume have C channels, one single 7x7 convolutional layers would contain $C \times (7 \times 7 \times C) = 49C^2$, while three 3x3 convolutional layers contain $3 \times C (3 \times 3 \times C) = 27C^2$.

3.3.5.2 Layer Sizing Patterns

The **input layer** (that contains the image) should be divisible by two many times. Common numbers include 32 (e.g. CIFAR-10), 64, 96 (e.g. STL-10), or 224 (e.g. common ImageNet ConvNets), 384, and 512.

The **conv layers** should be using small filters (e.g. 3x3 or at most 5x5), using a stride of $S=1$, and crucially, padding the input volume with zeros in such way that the conv layer does not alter the spatial dimensions of the input. Generally, $P=(F-1)/2$ can preserve the input size.

For the **pool layer**, the most common setting is to use max-pooling with 2x2 receptive fields, and with a stride of 2. Note that this discards exactly 75% of the activations in an input volume (due to downsampling by 2 in both width and height). There is a less common setting using 3x3 receptive field with a stride of 2. It is rare to see receptive field sizes for max pooling that are larger than three because the pooling is too lossy and aggressive, which usually leads to worse performance.

Padding can keep the spatial sizes constant after convolution, which improves performance. If the convolution layers perform valid convolutions without zero-padding, the size of the volumes would be reduced after each convolution and the information will be lost quickly.

4. Implementation

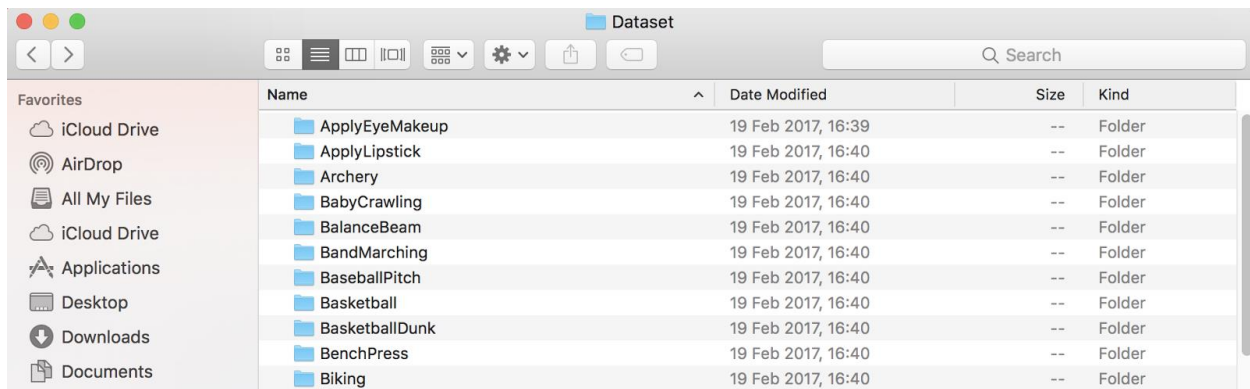
4.1 Data Processing

It is evident that Data Processing is the first step of Machine Learning. There are multiple purposes of processing the data. Firstly, we need to adapt the shape of the input data for the model. Besides, it is inefficient to train the model with high resolution since there is heavy computation to be done.

To process the data, the first task to be accomplished is getting familiar with the data.

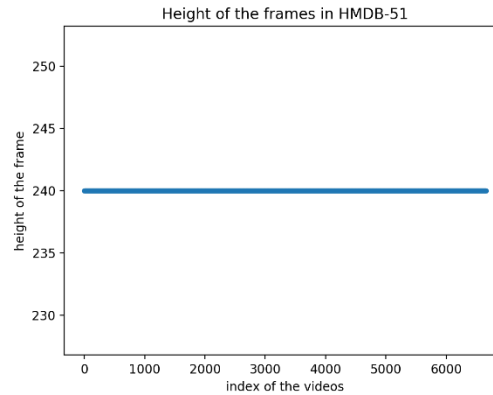
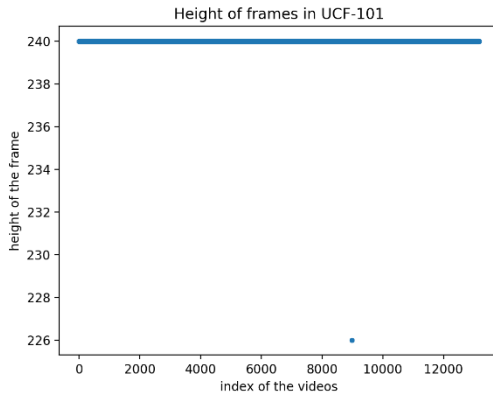
4.1.1 Resize Videos

Before applying the data to the network, the shape of input data has to be the same. In this occasion, the height of the frames, the width of the frames and the number of the frames have to be fixed given that we are working on 3D convolution. So the first step of data processing is to find out these three features of the videos in the dataset.

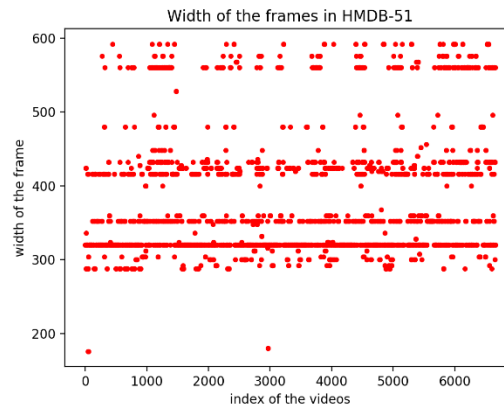
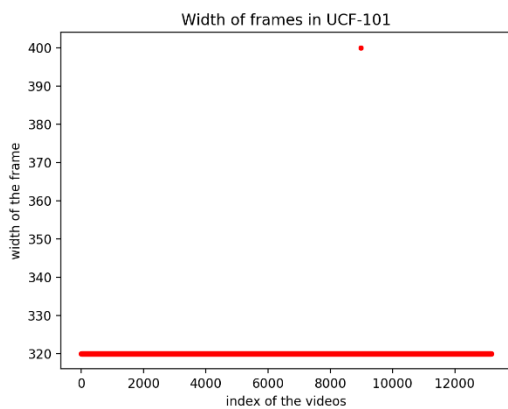


In our Dataset folder, there are numerous folders named by the label of the videos, which means we need a nested loop to traverse the whole dataset. The inner loop is used to access to the required information of each video in a fold, and the outer loop would change the folder being accessed.

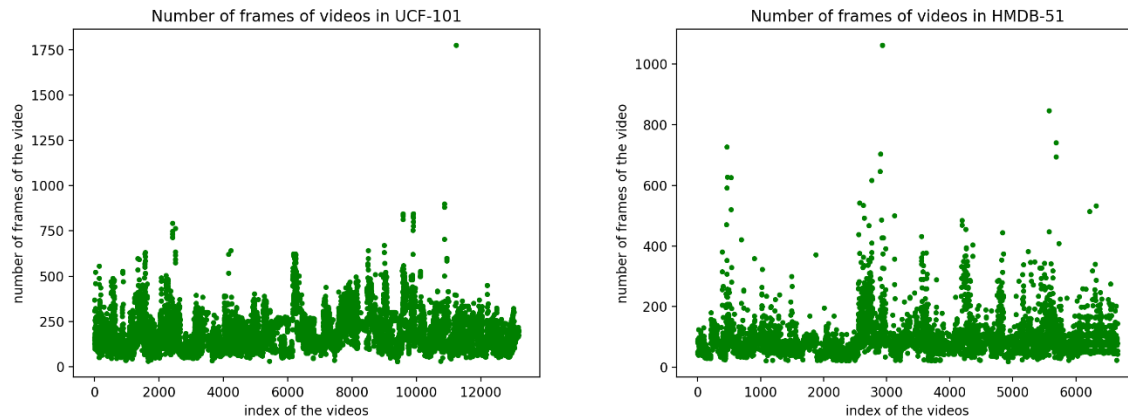
In class VideoCapture, there are three properties named CAP_PROP_FRAME_HEIGHT, CAP_PROP_FRAME_WIDTH and CAP_PROP_FRAME_COUNT representing the height of the frame, the width of the frame and the number of the frames respectively. By using the method get() in the class VideoCapture instead of counting the number of frames by another loop, the information we want to know can be efficiently accessed.



According to the graphs above, a majority of videos in UCF-101 share the same height, which is 240. There are only a few exceptions (4 videos, precisely) whose height does not equal to 240. Fortunately, all the videos in HMDB-51 have the same height, and the height is 240 too.



Regarding width, though UCF-101 has a few exceptions (4 videos) whose width does not equal 320, the rest of the videos have the same width. However, in HMDB-51, the situation is completely different. Most of the videos have a width equals to 320, but a significant part of the dataset does not share the same width, which is an inevitable problem in this stage.

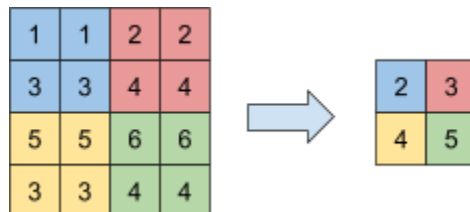


And there is a similar problem in the number of frames. In both datasets, the number of frames of videos is various. Unlike the situation of height, we cannot just abandon the exceptions.

Given that most videos in the dataset have 320*240 size of frames, I decided to resize all the videos to 320*240 using this algorithm: for those videos which have a smaller height or width, use black pixel (0,0,0 in RGB) to fill the frames; crop the central 320*240 pixels if the frame has a larger height or width. For the number of frames of videos, we noticed that a majority of videos (92.80%) has less than 300 frames. My idea was abandoning the redundant part if the video has more than 300 frames; if the video has less than 300 frames, then I can copy the last frame of that video and extend the duration of the video.

Unfortunately, when I use such a shape of data for my CNN, the Kill 9 signal was returned. In this occasion, I have to use a small size of frame otherwise the programme will be killed by the OS.

Instead of cropping a central window from the videos, I used another way to shrink the size of the frames, which might preserve more information in the videos.



For every 2-by-2 block in a frame, the average is computed and used as the volume of the pixel in the new frame. In this way, the size of frames was dramatically reduced to 180*120.

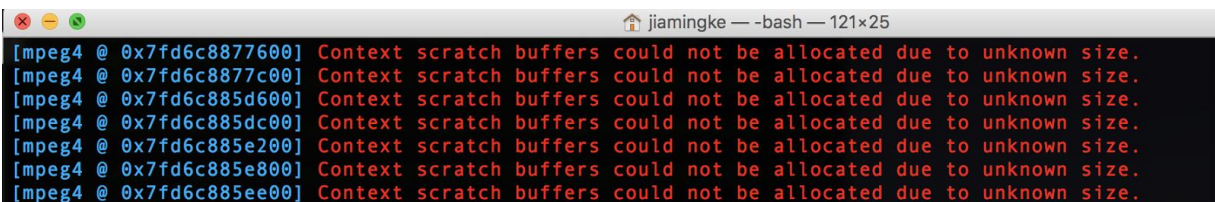
On the other hand, I also noticed that 300 frames are too much for an action. In many cases, the particular action is repeated and again in the video, which means, choosing a smaller number of frames might improve the efficiency of training and also does not affect the accuracy of classification. Eventually, I set 50 as a new threshold for the number of frames.

4.1.2 Write and Save the Videos

It seems to be a simple step in this project, but in OpenCV, there is no error message or warning if the saving has failed, which took me a long time on debugging.

1. Install OpenCV with ffmpeg support

In the beginning, I did not install OpenCV with ffmpeg support, which led to a situation that I could not save any video even though there was nothing wrong with the code.



```




[mp4 @ 0x7fd6c8877600] Context scratch buffers could not be allocated due to unknown size.
[mp4 @ 0x7fd6c8877c00] Context scratch buffers could not be allocated due to unknown size.
[mp4 @ 0x7fd6c885d600] Context scratch buffers could not be allocated due to unknown size.
[mp4 @ 0x7fd6c885dc00] Context scratch buffers could not be allocated due to unknown size.
[mp4 @ 0x7fd6c885e200] Context scratch buffers could not be allocated due to unknown size.
[mp4 @ 0x7fd6c885e800] Context scratch buffers could not be allocated due to unknown size.
[mp4 @ 0x7fd6c885ee00] Context scratch buffers could not be allocated due to unknown size.

```

2. Use a correct match of codec and file extension

While the function used to create videos file with OpenCV such as `cv2.VideoWriter`, `cv2.VideoWriter_fourcc` and `cv2.cv.FOURCC` are quite well documented^[20], what is not nearly as documented is the combination of codec and file extension required to write the video file successfully.^[21]

For example, I have the following combinations working on Mac: `*'mp4v' - '.mp4'` and `*'MJPG' - '.avi'`. In OpenCV, there is no any error messages or warning here if an incorrect combination is used, but we know if the saving has failed because of the size of the output video.

 testimage.jpg	21 Feb 2017, 16:30	18 KB	JPEG image
 example.avi	Today, 17:38	6 KB	AVI movie
 t.avi	8 Dec 2016, 14:15	125 KB	AVI movie

Now we can focus on the main part of the video writing implementation.

```

25     while True:
26         ret, frame = cap.read()
27         if frame is None:
28             break
29         if writer is None:
30             fps = cap.get(cv2.CAP_PROP_FPS)
31             filename = "Dataset_2/" + label + "/" + video
32             print("Processing: " + filename)
33             writer = cv2.VideoWriter(filename, fourcc, int(fps), (160, 120), True)
34             output = np.zeros((120, 160, 3), dtype="uint8")
35             for x in range(0, 120):
36                 for y in range(0, 160):
37                     temp = np.array([[frame[x*2, y*2], frame[x*2, y*2+1],
38                                     frame[x*2+1, y*2], frame[x*2+1, y*2+1]]])
39                     a = np.mean(temp, axis = 0)
40                     a = np.array(a, dtype = "uint8")
41                     output[x, y] = a
42             writer.write(output)
43         cap.release()
44         writer.release()

```

Line 25 to 42 is a loop for saving frames to a particular output file. From line 35 to line 41, it implements the resize of the frame I mentioned before, and the size reduces to 160*120 after that nested loop.

We also notice that a VideoWriter is instantiated in line 33 and an empty frame is constructed in line 34. According to the definition of the empty frame, the shape of the frames is (height, width, the number of channels). However, in line 33, the shape is (width, height) instead of (height, width). We have to be careful about it otherwise we could not successfully save the video to the disk.

4.1.3 Shrink the Dataset

Initially, I decided to train my model for both UCF-101 and HMDB-51. There are 20,034 videos on the total. I attempt to use cuDNN for acceleration. However, the version of CUDA installed in Linux Lab is 7.5 while latest TensorFlow requires CUDA 8.0. Without the help of GPU acceleration, I considered about applying for the privilege to use the supercomputer in the university. Unfortunately, I was told that I had to take a training session about supercomputer before I can use it but I did not have enough time for the course. Then I asked my friend for using his GTX 1070 GPU to run the programme. After spending two days on tools installation, I finally run the programme in his machine, but the programme required more than 8G GPU memory, which was already the limit of my friend's machine.

```

W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn
't compiled to use AVX2 instructions, but these are available on your machine a
nd could speed up CPU computations.
W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn
't compiled to use FMA instructions, but these are available on your machine an
d could speed up CPU computations.
8/80 [==>.....] - ETA: 8783s - loss: 4.1165 - acc: 0.000
0e+00

```

Without the help of GPU or supercomputer, a 50-frame video takes more than 100 seconds in the training process in a 2.9GHz quad-core Intel Core i7 processor. Given that there are 20,034 videos in the dataset, and 80% of them are used for training, the training process takes around 1,602,700 seconds for each epoch. Take ten epochs for example, at least 16,027,000 seconds (4452 hours, 185.5 Days) are required.

I reported the problem and described the situation to my supervisor. After our discussion, we decided to choose five actions (includes about 600 videos) from UCF-101 for training and testing. Mathematically, there are around 480 videos needed to be trained. Assume each video takes 100 seconds in training process, then 48,000 seconds are required in each epoch. Training the model for ten epochs requires 480,000 seconds (133.3 hours, 5.6 Days), which is more acceptable than before.

I chose five sports from the UCF-101 dataset and named the new dataset as UCFSprot-5. That five chosen sports, including Archery, Biking, Hulahoop, Push Ups and Rope Climbing, are relatively easy to be distinguished from each other. Plus, in the video of each sport, there is only one human in a video. Under such conditions, hopefully, the accuracy of the model can be higher so that the result would be more reliable.

4.1.4 Dataset API

In this part, there are three functions I need to implement: label the data; randomly generate training and testing set; load data and labels to memory.

1. Labeling

```
5 def load_list():
6
7     rootpath = os.path.abspath(os.path.dirname(__file__))
8     os.chdir(rootpath)
9     labellist = os.listdir('UCFSport5')
10    labellist = labellist[1:]
11
12    i = 0
13    labels = [None] * 200
14    for label in labellist:
15        labels[i] = labellist[i]
16        i += 1
17    labels = labels[:i]
18    return labels
```

```
20 def load_dict():
21     rootpath = os.path.abspath(os.path.dirname(__file__))
22     os.chdir(rootpath)
23     labellist = os.listdir('UCFSport5')
24     labellist = labellist[1:]
25
26     i = 0
27     labels = {}
28     for label in labellist:
29         labels[labellist[i]] = i
30         i += 1
31     return labels
32
```

I defined two methods: `load_list()` and `load_dict()`, which means load the labels into a list and a dict respectively. Given that the label is the name of the folder, I can simply invoke `os.listdir()` to get all the names of the folders in the dataset.

2. Random Split

```
14 i = 0
15 rootpath = os.path.abspath(os.path.dirname(__file__))
16 labels = data_labels.load_dict()
17 list = data_labels.load_list()
18
```

Invoke the two methods defined above, and store the labels in variable *labels* and *list* with the different data structure.

```
20
21 for label in list:
22     os.chdir(rootpath)
23     path = 'UCFSport5/' + label
24     os.chdir(path)
25     list_in_label = os.listdir()
26     list_in_label = list_in_label[1:]
27     for l in list_in_label:
28         x[i] = 'UCFSport5/' + label + '/' + l
29         y[i] = labels.get(label)
30         i += 1
31
32 x = x[:i]
33 y = y[:i]
```

Then I used a nested loop to traverse all the videos in the dataset and store their paths and labels in x and y respectively.

```
35 perm = np.random.permutation(i)
36 trainlist = perm[:int(i*train_ratio)]
37 testlist = perm[int(i*train_ratio):]
38
```

Invoke NumPy.random.permutation() to generate a random permutation and divide the array into two parts.

```
40 file = open('train_list_x.txt', 'w')
41 for j in trainlist:
42     file.write(x[j] + '\n')
43 file.close()
44
```

Create four different files and save the training and testing set to the disk using the code similar to the one shown above. Name that 4 files as train_list_x, train_list_y, test_list_x and test_list_y.

3. Loading

```
39 i = 0
40 file = open('test_list_y.txt', 'r')
41 for line in file:
42     str = line[:len(line)-1]
43     y_test[i] = str
44     i += 1
45 y_test = y_test[:i]
46 file.close()
47
48 return ((x_train, y_train), (x_test, y_test))
```

Finally, we can open the files we saved by passing different names to the `open()` method, read the file line by line and load all the contents of that four files into memory, return ((Training_data, Training_labels), (Testing_data, Testing_label)).

4.2 Neural Network

In this section, I will focus on introducing how to use Keras for building a Neural Network and then will also explain how I use it to construct my own CNN. Finally, I will mention a few configuration of my CNN that I have attempted for improving the accuracy of the model.

4.2.1 Relevant Classes in Keras

I will explain the essential components of Keras including Models and Layers. Also, the rest components, i.e. Losses, Metrics, Optimizers and Visualisation will be briefly introduced as well.

1. Models

The Sequential Model and the Model class used with functional API are the only two types of models available in Keras. In Keras Documentation^[15], it shows that those two models have some methods in common which would be used in the project.

- `model.summary()`: prints a summary representation of your model.
- `model.save_weights(file path)`: saves the weights of the model as an HDF5 file, which requires the `h5py` module.
- `model.load_weights(filepath, by_name=False)`: loads the weights of the model from a HDF5 file (created by `save_weights`). By default, the architecture is expected to be unchanged. To load weights into a different architecture (with some layers in common), use `by_name=True` to load only those layers with the same name.

Since I used the Sequential model only, the other model will not be introduced here. You can visit Keras Documentation for more details.

The Sequential model is a linear stack of layers, it can be created by passing a list of layer instances to the constructor, or simply add layers via the `.add()` method:

```

19 model = Sequential()
20
21 model.add(ZeroPadding3D((1,1,1), input_shape=data_shape))
22 model.add(Convolution3D(32, 3, 3, 3, activation = 'relu'))
23 model.add(BatchNormalization(axis=1, momentum=0.99, epsilon=0.001))
24 model.add(ZeroPadding3D((1,1,1)))
25 model.add(Convolution3D(32, 3, 3, 3, activation = 'relu'))
26 model.add(BatchNormalization(axis=1, momentum=0.99, epsilon=0.001))
27 model.add(MaxPooling3D(pool_size=(2, 2, 2), strides=(2,2,2)))
28

```

The model needs to know what input shape it should expect. For this reason, the first layer in a Sequential model needs to receive information about its input shape, then the following layers can make automatic shape inference. More details about it will be explained later.

Before training a model, we need to configure the learning process, which is done via the `.compile()` method that receives three arguments: an optimizer, a loss function and a list of metrics.

- An optimizer - it could be a string identifier (such as `rmsprop` or `adagrad`), or an instance of the `Optimizer` class, which will be introduced later.
- A loss function - this is the objective that the model will try to minimise. Theoretically, the accuracy will be increased as the loss is decreasing. A loss function can be a string identifier of existing loss function (e.g. `categorical_crossentropy`) or an objective function. More details will be shown later.
- A list of metrics - for any classification problem, it can be set as `metrics=['accuracy']`, which will display the accuracy of the model during training and testing.

After compiling, a `.fit()` method needs to be invoked for starting the training process.

In Keras, a `.fit()` method is used to train the model for a fixed number of epochs and return a `History` object, whose attribute (`History.history`) is a record of training loss values and metrics values at successive epochs, as well as validation loss values and validation metrics values.

2. Layers

In this part, I will introduce the Layers I used in my model including Core Layers, Convolutional Layers, Pooling Layers and Normalisation Layers.

- **Core Layers**

Dense - the regular densely-connected NN layer.

It implements the operation: $output = activation(dot(input, kernel) + bias)$ where activation is the element-wise activation function passed as the activation argument, the kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer (only applicable if use_bias is True).

Activation - the activation function to the output.

To use a particular activation function, directly apply the name of the function to the layer. Some activation functions are already implemented by Keras, which includes elu, softplus, softsign, relu, tanh, sigmoid, hard_sigmoid and linear. Some of the activations are already explained in Chapter 3.

Dropout - it consists of randomly setting a fraction rate of input units to 0 at each update during training time, which helps prevent overfitting.^[22]

Flatten - flattens the data.

- **Convolutional Layers**

Conv3D - 3D convolution layer (e.g. spatial convolution over volumes).

This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. When using this layer as the first layer in a model, provide the key argument input_shape (tuple of integers, does not include the sample axis), e.g. input_shape=(128,128,128,3) for 128x128x128 volumes with a single channel.

ZeroPadding3D - Zero-padding layer for 3D data (spatial or spatio-temporal).

- **Pooling Layers**

MaxPooling3D - Max pooling operation for 3D data (spatial or spatiotemporal).

- **Normalization Layers**

BatchNormalization - Batch normalization layer^[19].

It normalises the activations of the previous layer at each batch, i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1.

3. Losses

A loss function (a.k.a. objective function or optimisation score function) is one of the two parameters required to compile a model:

```
56 model.compile(loss='categorical_crossentropy',  
57               optimizer=sgd,  
58               metrics=['accuracy'])  
59
```

This method can either be passed the name of an existing loss function, or a TensorFlow/Theano symbolic function that takes two arguments (true labels and predictions) and returns a scalar for each data-point. Besides, the optimised objective is the mean of the output array across all data-points.

There are many different loss functions available in Keras, which can be checked in the [Documentation](#)^[15] for more details. When using `categorical_crossentropy` loss, your targets should be in the categorical format (e.g. if you have ten classes, the target for each sample should be a 10-dimensional vector that is all zeros except a one at the index corresponding to the class of the sample).

To convert integer targets into categorical targets, you can use the Keras utility `to_categorical()`:

```
95 # Convert class vectors to binary class matrices.  
96 Y_train_s = np_utils.to_categorical(y_train_s, nb_classes)  
97 Y_test_s = np_utils.to_categorical(y_test_s, nb_classes)  
98
```

4. Optimizers

The Optimizer is another argument required for compiling a Keras model. There are two ways to set the optimizer in a `.compile()` method. You can either instantiate an optimizer before passing it to `model.compile()` or simply call it by its name. In the latter case, the default parameter for the optimizer will be used.

There are some available optimizers in Keras. Personally, I have tested the performance of the model while using SGD and RMSprop. Stochastic gradient descent (SGD) optimizer includes support for momentum, learning

rate decay and Nesterov momentum. SGD has proved to be an effective way of training deep networks, and SGD variant such as momentum^[23] and Adagrad^[24] have been used to achieve state of the art performance.^[19]

5. Metrics

A metric is a function used to judge the performance of a model. Metric functions are to be supplied in the metrics parameter when a model is compiled.

Similar to the loss function, it can accept the name of an existing metric or a TensorFlow/Theano symbolic function. The only difference is that the result of evaluating a metric are not used when training the model.

4.2.2 Codes and Comments

In this part, I will show part of the python files which are relevant to the neural network, and explain the code of mine. There are three files showing below. The first file is the model file containing all the details about the model, and it will invoke some self-defined methods in the second and the third files. The second file is a python script accessing to our dataset and read the train and test files as well as the correspondent labels. The last one is used to load the videos in the Dataset frame by frame. Since there are too many videos in the dataset and they require large memory while the model is training, I have to divide the dataset into several portions and train the model with them.

1. Model Definition

```
19 model = Sequential()
20
21 model.add(ZeroPadding3D((1,1,1), input_shape=data_shape))
22 model.add(Convolution3D(32, 3, 3, 3, activation = 'relu'))
23 model.add(BatchNormalization(axis=1, momentum=0.99, epsilon=0.001))
24 model.add(ZeroPadding3D((1,1,1)))
25 model.add(Convolution3D(32, 3, 3, 3, activation = 'relu'))
26 model.add(BatchNormalization(axis=1, momentum=0.99, epsilon=0.001))
27 model.add(MaxPooling3D(pool_size=(2, 2, 2), strides=(2,2,2)))
28
```

The figure above shows how I instantiate a Sequential model and add the first two Convolutional layers to it. For that two layer, I set ReLu as their activation function, and I also add a Normalisation layer for each of them.

```
29 model.add(ZeroPadding3D((1,1,1)))
30 model.add(Convolution3D(64, 3, 3, 3, activation = 'relu'))
31 model.add(BatchNormalization(axis=1, momentum=0.99, epsilon=0.001))
32 model.add(ZeroPadding3D((1,1,1)))
33 model.add(Convolution3D(64, 3, 3, 3, activation = 'relu'))
34 model.add(BatchNormalization(axis=1, momentum=0.99, epsilon=0.001))
35 model.add(MaxPooling3D(pool_size=(2, 2, 2), strides=(2,2,2)))
36
```

Similar to the previous one, here are another two Convolutional layers. The only difference they have is that there are 32 filters in each of the first two Convolutional layers while the second two Convolutional layers have 64 filters in each of them.

```
37 model.add(ZeroPadding3D((1,1,1)))
38 model.add(Convolution3D(128, 3, 3, 3, activation = 'relu'))
39 model.add(BatchNormalization(axis=1, momentum=0.99, epsilon=0.001))
40 model.add(ZeroPadding3D((1,1,1)))
41 model.add(Convolution3D(128, 3, 3, 3, activation = 'relu'))
42 model.add(BatchNormalization(axis=1, momentum=0.99, epsilon=0.001))
43 model.add(ZeroPadding3D((1,1,1)))
44 model.add(Convolution3D(128, 3, 3, 3, activation = 'relu'))
45 model.add(BatchNormalization(axis=1, momentum=0.99, epsilon=0.001))
46 model.add(MaxPooling3D(pool_size=(2, 2, 2), strides=(2,2,2)))
47
```

In the third part, there are more Convolutional layers and more filters in each of them. Except that, there is no difference between this part and the previous part.

```
48 model.add(Flatten())
49 model.add(Dense(512, activation = 'relu'))
50 model.add(Dropout(0.5))
51 model.add(Dense(nb_classes))
52 model.add(Activation('softmax'))
53
54 # Let's train the model using SGD
55 sgd = SGD(lr=0.0001, decay=1e-6, momentum=0.9, nesterov=True)
56 model.compile(loss='categorical_crossentropy',
57               optimizer=sgd,
58               metrics=['accuracy'])
59
```

In the final part of my model, I used two regular densely-connected layers. I chose ReLu as the activation function for the first Dense layer and add a Dropout to it in the case of overfitting.^[22] For the second one, I used the number of classes as the number of the filter, and chose softmax as the activation function, which is the prevalent style of designing the CNN.

After that, I instantiate an SGD with 0.0001 learning rate, 1e-6 for decay and 0.9 for momentum, and use it as the optimizer in .compile() method.

```

64 nb_piece = int(len(x_train)*1.0/piece_size)
65 if len(x_train)%piece_size != 0:
66     nb_piece += 1
67
68 for i in range(0,nb_piece):
69     if (i+1)*piece_size > len(x_train) or (i+1)*(piece_size/4) > len(x_test):
70         x_train_s = x_train[i*piece_size:]
71         y_train_s = y_train[i*piece_size:]
72         x_test_s = x_test[i*(piece_size/4):]
73         y_test_s = y_test[i*(piece_size/4):]
74         X_train_s = np.zeros((len(x_train_s), 50, 120, 160, 3),dtype = "uint8")
75         X_test_s = np.zeros((len(x_test_s), 50, 120, 160, 3),dtype = "uint8")
76     else:
77         x_train_s = x_train[i*piece_size:(i+1)*piece_size]
78         y_train_s = y_train[i*piece_size:(i+1)*piece_size]
79         x_test_s = x_test[i*int(piece_size/4):(i+1)*int(piece_size/4)]
80         y_test_s = y_test[i*int(piece_size/4):(i+1)*int(piece_size/4)]
81         X_train_s = np.zeros((piece_size, 50, 120, 160, 3),dtype = "uint8")
82         X_test_s = np.zeros((int(piece_size/4), 50, 120, 160, 3),dtype = "uint8")
83
84     j = 0
85     for video in x_train_s:
86         X_train_s[j] = video_reader.load(video)
87         j += 1
88
89
90     j = 0
91     for video in x_test_s:
92         X_test_s[j] = video_reader.load(video)
93         j += 1
94
95     # Convert class vectors to binary class matrices.
96     Y_train_s = np_utils.to_categorical(y_train_s, nb_classes)
97     Y_test_s = np_utils.to_categorical(y_test_s, nb_classes)
98
99
100     X_train_s = X_train_s.astype('float32')
101     X_test_s = X_test_s.astype('float32')
102     X_train_s /= 255
103     X_test_s /= 255
104
105     model.fit(X_train_s, Y_train_s,
106             batch_size=batch_size,
107             nb_epoch=nb_epoch,
108             validation_data=(X_test_s, Y_test_s),
109             shuffle=True)
110
111
112     model.save('model.h5')
113     model.save_weights('model_weight.h5')
114

```

In this part of code, I divided a whole dataset into several pieces and used the outer loop to traverse them and train the model with them. From line 85 to 93, I used self-defined method `video_reader.load()` to load the videos from the disk. Then, I convert the class vectors to binary class matrices in line 96 and 97, which is required by loss function `categorical_crossentropy`.

2. Dataset API

```

39     i = 0
40     file = open('test_list_y.txt','r')
41     for line in file:
42         str = line[:len(line)-1]
43         y_test[i] = str
44         i += 1
45     y_test = y_test[:i]
46     file.close()
47
48     return ((x_train,y_train),(x_test,y_test))

```


There are another three parts similar to the one shown above. This method will open 4 text files named test_list_x.txt, test_list_y.txt, train_list_x.txt as well as train_list_y.txt and save their contents to 4 variables called x_test, y_test, x_train as well as y_train, then those 4 variable would be return.

3. Video Reader Definition

```
1 #This file contains a method to load the video
2
3 import numpy as np
4 import cv2
5
6 def load(str):
7     cap = cv2.VideoCapture(str)
8     video = np.zeros((50, 120, 160, 3), dtype="uint8")
9     current_f = 0
10
11     while True:
12         retv, frame = cap.read()
13
14         if retv is not True:
15             break
16
17         video[current_f] = frame
18         current_f += 1
19
20     cap.release()
21
22     return video
23
```

In line 7, method VideoCapture is invoked to instantiate a VideoCapture object with the given path as a parameter, the video referred by the path str will be opened. In the following lines, I preallocate some memory for a video and initialise a variable current_f which is used to count the index of the current frame.

The While loop from line 11 to line 18 implement a function that if the next frame of the video is successfully read, the frame will be stored in the preallocated memory; otherwise, the loop would be a break.

4.2.3 Attempted Configuration

In this section, I would like to talk about the configuration of the network I tried, but the results of that configuration and the analysis would be provided in Chapter 5 and Chapter 6 respectively.

```

18 model = Sequential()
19
20 model.add(Convolution3D(32, 3, 3, 3, border_mode='same',
21                        input_shape=data_shape))
22 model.add(Activation('relu'))
23 model.add(MaxPooling3D(pool_size=(2, 2, 2)))
24
25 model.add(Convolution3D(64, 3, 3, 3, border_mode='same'))
26 model.add(Activation('relu'))
27 model.add(MaxPooling3D(pool_size=(2, 2, 2)))
28
29 model.add(Convolution3D(128, 3, 3, 3, border_mode='same'))
30 model.add(Activation('relu'))
31 model.add(MaxPooling3D(pool_size=(2, 2, 2)))
32 model.add(Dropout(0.5))
33
34 model.add(Flatten())
35 model.add(Dense(nb_classes))
36 model.add(Activation('softmax'))
37
38 sgd = SGD(lr=0.0001, decay=1e-6, momentum=0.9, nesterov=True)
39 model.compile(loss='categorical_crossentropy',
40              optimizer=sgd,
41              metrics=['accuracy'])
42

```

The code above was my initial network; there were four layers on total; 32 filters in the first Convolutional layer, 64 filters in the second one and 128 filters in the third one. For the densely-connected layer, I set the number of the classes as the number of the filters, which is 101 for the UCF-101 dataset and 51 for the HMDB-51 dataset.

The network I show below is similar to my current network, except it is not as deep as my current one and it used RMSprop as the optimizer.

```

33 model = Sequential()
34
35 model.add(Convolution3D(32, 3, 3, 3, border_mode='same',
36                        input_shape=X_train_s.shape[1:]))
37 model.add(Activation('relu'))
38 model.add(Convolution3D(32, 3, 3, 3))
39 model.add(Activation('relu'))
40 model.add(MaxPooling3D(pool_size=(2, 2, 2)))
41 model.add(Dropout(0.25))
42
43 model.add(Convolution3D(64, 3, 3, 3, border_mode='same'))
44 model.add(Activation('relu'))
45 model.add(Convolution3D(64, 3, 3, 3))
46 model.add(Activation('relu'))
47 model.add(MaxPooling3D(pool_size=(2, 2, 2)))
48 model.add(Dropout(0.25))
49
50 model.add(Flatten())
51 model.add(Dense(512))
52 model.add(Activation('relu'))
53 model.add(Dropout(0.5))
54 model.add(Dense(nb_classes))
55 model.add(Activation('softmax'))
56
57 # Let's train the model using RMSprop
58 model.compile(loss='categorical_crossentropy',
59              optimizer='rmsprop',
60              metrics=['accuracy'])
61

```

4.3 Object Detection and Object Tracking

Initially, my idea was using Object Detection for every frame then I can modify the object I found in that frame. However, purely use object detection will always lead to failure in finding objects, which is the reason that I started researching and implementing Object Tracking. Then I decided to apply Object Detection in the first frame and use Object Tracking for the rest. It is a good idea regarding processing speed, but Object Tracking would lose track of the object occasionally.

Given that it is no need to be real-time tracking in this project, I come up with the idea that applying Object Detection for every frame and use Object Tracking if the Object Detection classifier fails to find the object. With enough computation time, it is a good idea that maximising the accuracy of finding the object. However, none of the algorithms above can solve the problem that the detection failure happens in the beginning. The most promising way to prevent that is manually select the ROI in the first frame, which is apparently not the most efficient way.

In this section, I will introduce two different classifier for Object Detection and two different algorithms for Object Tracking.

4.3.1 Haar cascade

It is a Machine Learning approach for visual object detection which is capable of processing images extremely rapidly and achieving high detection rates.^[26] Three key contributions distinguish it:

- a new image representation called the “Integral Image” which allows the features used by our detector to be computed quickly.^[53]
- a learning algorithm based on AdaBoost, which selects a small number of critical visual features from a larger set and yields extremely efficient classifiers^[27].
- a method for combining more complex classifiers increasingly in a “cascade” which allows background regions of the image to be quickly discarded while spending more computation on promising object-like regions.^[53]

Now I will introduce the usage of Haar Cascade in OpenCV and the instruction of training our own Haar Cascade.

1. Usage in OpenCV

Let’s assume that we need to detection some common objects (e.g. faces, eyes) whose cascade can be found on the Internet. Then we can download the cascade from the Internet.

Assume faces are the objects we want to detect and we have already downloaded the face-cascade to the working directory.

```

14 #https://github.com/Itseez/opencv/blob/master/data/haarcascades/haarcascade_frontalface_default.xml
15 face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
16

```

Simply use `.CascadeClassifier()` to load a classifier from file.

```

20 while 1:
21     ret, img = cap.read()
22     if img is None:
23         break
24     gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
25     faces = face_cascade.detectMultiScale(gray, 1.3, 5)
26

```

Then, convert the current frame into a grey image and apply the grey image to the `.detectMultiScale()` method in the classifier.

```

27     for (x,y,w,h) in faces:
28         cv2.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 2)
29
30
31     cv2.imshow('img',img)

```

After that, we use an inner loop to draw rectangles on all the faces we found in the frame and display it, then we will be able to see the result.

2. Train a Haar Cascade

In fact, we cannot always find the cascade we want on the Internet. Sometimes if we need to detect the object which is not that prevalent, for example, a sword for Fencing or a hula hoop, we might need to train our cascade for object detection.

Firstly, set up the tools we need for cascade training.

1. OpenCV

```

sudo apt-get install git
git clone https://github.com/Itseez/opencv.git

```

2. Compiler

```

sudo apt-get install build-essential

```

3. Libraries

```

sudo apt-get install cmake git libgtk2.0-dev pkg-config libavcodec-dev
libavformat-dev libswscale-dev

```

4. Python bindings and such

```

sudo apt-get install python-dev python-numpy libtbb2 libtbb-dev
libjpeg-dev libpng-dev libtiff-dev libjasper-dev libdc1394-22-dev

```

5. OpenCV development library

`sudo apt-get install libopencv-dev`

Then we need positive data and negative data for building a Haar Cascade. We can either use our images or download the picture from ImageNet for positive data and negative data. Besides, we also need to specify ROI where the object is.

```
13 def store_raw_images():
14     neg_images_link = '//image-net.org/api/text/imagenet.synset.geturls?wnid=n00523513'
15     neg_image_urls = urllib.request.urlopen(neg_images_link).read().decode()
16     pic_num = 1
17
18     if not os.path.exists('neg'):
19         os.makedirs('neg')
20
21     for i in neg_image_urls.split('\n'):
22         try:
23             print(i)
24             urllib.request.urlretrieve(i, "neg/"+str(pic_num)+".jpg")
25             img = cv2.imread("neg/"+str(pic_num)+".jpg",cv2.IMREAD_GRAYSCALE)
26             # should be larger than samples / pos pic (so we can place our image on it)
27             resized_image = cv2.resize(img, (100, 100))
28             cv2.imwrite("neg/"+str(pic_num)+".jpg",resized_image)
29             pic_num += 1
30
```

Let's assume we want to download the data from ImageNet. The script above can visit the links, collect the URLs and proceed to visit them. Then it will convert the images to grayscale, resize them and then save.

```
9 def find_uglies():
10     match = False
11     for file_type in ['neg']:
12         for img in os.listdir(file_type):
13             for ugly in os.listdir('uglies'):
14                 try:
15                     current_image_path = str(file_type)+'/'+str(img)
16                     ugly = cv2.imread('uglies/'+str(ugly))
17                     question = cv2.imread(current_image_path)
18                     if ugly.shape == question.shape and
19                        not(np.bitwise_xor(ugly,question).any()):
20
21                         print(current_image_path)
22                         os.remove(current_image_path)
```

However, there are some images in ImageNet which are not available anymore, using the script above can remove those blank images. After that, we can generate the description files for negative images and positive images.

```
27 def create_pos_n_neg():
28     for file_type in ['neg']:
29
30         for img in os.listdir(file_type):
31
32             if file_type == 'pos':
33                 line = file_type+'/'+img+' 1 0 0 50 50\n'
34                 with open('info.dat','a') as f:
35                     f.write(line)
36             elif file_type == 'neg':
37                 line = file_type+'/'+img+'\n'
38                 with open('bg.txt','a') as f:
39                     f.write(line)
40
```

Now that we have positive data and negative data, we need to run the following command via Terminal to create the vector file, which is where we stitch all of our positive images together.

```
opencv_createsamples -info info/info.lst -num 1950 -w 20 -h 20 -vec
```

Then we have our vector file named positives.vec, so we can run the following command to train a cascade:

```
opencv_traincascade -data data -vec positives.vec -bg bg.txt -numPos 1800 -numNeg 900 -numStages 10 -w 20 -h 20
```

4.3.2 YouTube-BoundingBoxes

It is a large-scale dataset of video URLs with densely-quality single-object bounding box annotations, and it is developed by a group of researchers and engineers at Google Research which is interested in helping to advance the field of machine learning related to video understanding.

The data set consists of approximately 380,000 15-20s video segments extracted from 240,000 different publicly visible ng videos, automatically selected to feature objects in natural settings without editing or post-processing, with a recording quality often akin to that of a hand-held cell phone camera. All these video segments were human-annotated with high precision classifications and bounding boxes at one frame per second.^[54]

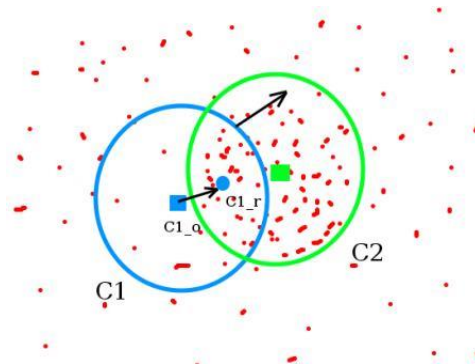
However, at most one object can be tracked for each video segment, but multiple segments could be extracted from the same video, which means that the number of ids is not enough for identifying video segments uniquely. Besides, there are only 23 classes in this classifier, which is still not enough for either UCF-101 or HMDB-51.

4.3.3 Meanshift and CAMshift

In this section, I will introduce the algorithms of Meanshift and Camshift and then explain about how to use Meanshift and Camshift in OpenCV for Object Tracking.

1. Meanshift

Given a small window, we have to move the window to the area of maximum pixel density (or maximum number of points), which is the idea of Meanshift.^[28] It is illustrated in a simple image given below:



The initial window is shown in a blue circle with the name “C1”. Its starting centre is marked in blue rectangle, named “C1_o”. And we search for the centroid of points inside that window called “C1_r” (marked in a small blue circle) which is the real centroid of the window. Then move the window such that circle of the new window matches with the previous centroid, again find the new centroid. Continue the iterations until centre of the window and its centroid falls on the same location (or with a small desired error). Then what you obtain is a window with maximum pixel distribution. It is marked with a green circle in the image.

```
13 # set up the ROI for tracking
14 roi = frame[r:r+h, c:c+w]
15 hsv_roi = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
16 mask = cv2.inRange(hsv_roi, np.array((0., 60., 32.)), np.array((180., 255., 255.)))
17 roi_hist = cv2.calcHist([hsv_roi],[0],mask,[180],[0,180])
18 cv2.normalize(roi_hist,roi_hist,0,255,cv2.NORM_MINMAX)
19
```

Set up the target, find the histogram so that we can back-project the target on each frame for calculation of meanshift. Besides, use `.inRange()` to discard the false values due to low light.

```
20 # Setup the termination criteria, either 10 iteration or move by atleast 1 pt
21 term_crit = ( cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 1 )
22
23 while(1):
24     ret ,frame = cap.read()
25
26     if ret == True:
27         hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
28         dst = cv2.calcBackProject([hsv],[0],roi_hist,[0,180],1)
29
30         # apply meanshift to get the new location
31         ret, track_window = cv2.meanShift(dst, track_window, term_crit)
32
33         # Draw it on image
34         x,y,w,h = track_window
35         img2 = cv2.rectangle(frame, (x,y), (x+w,y+h), 255,2)
36         cv2.imshow('img2',img2)
```

Setup the termination criteria before the loop, then the `.meanShift()` method can be invoked within the loop for getting the new location. Simply use `.rectangle()` to visualise the result.

2. CAMshift

There is a drawback for the meanshift: the window always has the same size no matter how far the object is away from the camera or close to the camera. In 1988, Continuously Adaptive Meanshift (CAMshift) was published to solve the problem mentioned above.^[29]

It applies meanshift first. Once meanshift converges, it updates the size of the window as, $s = 2 \times \sqrt{\frac{M_{00}}{256}}$. It also calculates the orientation of best fitting ellipse to it. Again it applies the meanshift with new scaled search window and previous window location. The process is continued until required accuracy is met.^[28]

It is similar to meanshift, but it returns a rotated rectangle and box parameters (used to be passed as a search window in next iteration).

5. Results

Since my idea is to modify the video and evaluate the influence on classification, my experiment method is to train a CNN first, then apply the unmodified testing data to the network and get the prediction result. With that results, which are supposed to be a Numpy array[15], we can use PCA to convert the high dimensional data into two dimensional data and visualise the result. Theoretically, we can see 5 cluster centres. Then we can apply the modified videos to the model and plot the result on the same chart. Therefore we can see the influence of the modification.

5.1 Data Pre-processing

Now we can focus on the difference of image size of the videos before and after pre-processing:



The image on the left is the original video from UCF-101, whose frame size is 360x240. I attempted to apply the videos with such a frame size to the neural network, but it consumed too many memories then the programme was killed by the Operating System because the OS received a “Killed: 9” signal which cannot be ignored.

At that time, I did not know too much about Convolutional Neural Network, so I did not have any other solution but to resize the data I used. Eventually, I used 50x120x160 as the shape of my data, which was 300x240x320 in the previous attempt, and the new shape worked well in my neural network.

5.2 Model Summary and Classification Performance

5.2.1 Summary

Layer (type)	Output Shape	Param #
zero_padding3d_1 (ZeroPadding3D)	(None, 52, 122, 162, 3)	0
conv3d_1 (Conv3D)	(None, 50, 120, 160, 32)	2624
batch_normalization_1 (Batch Normalization)	(None, 50, 120, 160, 32)	200
zero_padding3d_2 (ZeroPadding3D)	(None, 52, 122, 162, 32)	0
conv3d_2 (Conv3D)	(None, 50, 120, 160, 32)	27680
batch_normalization_2 (Batch Normalization)	(None, 50, 120, 160, 32)	200
max_pooling3d_1 (MaxPooling3D)	(None, 25, 60, 80, 32)	0

The chart above is the summary of the first two convolutional layers of my model. Just like I mentioned above, my input shape was 50x120x160. In my neural network, I set zero-padding $P = 1$, stride $S = 1$, receptive field size $F = 3$. As I explained in 3.3.4, the input volume and output volume had the same size spatially, which perfectly matches the result above.

The situation in the second layer was the same as the first one except for the number of the input data channels. The original input data had three channel (RGB). After the first convolutional layer whose depth was 32, the number of input data channels of the second layer became 32 channels.

After the first max pooling layer, 75% of the data were discarded, so the shape of the max pooling output was 25x60x80.

zero_padding3d_3 (ZeroPadding3D)	(None, 27, 62, 82, 32)	0
conv3d_3 (Conv3D)	(None, 25, 60, 80, 64)	55360
batch_normalization_3 (Batch Normalization)	(None, 25, 60, 80, 64)	100
zero_padding3d_4 (ZeroPadding3D)	(None, 27, 62, 82, 64)	0
conv3d_4 (Conv3D)	(None, 25, 60, 80, 64)	110656
batch_normalization_4 (Batch Normalization)	(None, 25, 60, 80, 64)	100
max_pooling3d_2 (MaxPooling3D)	(None, 12, 30, 40, 64)	0

In the third and fourth convolutional layers, the situation was similar to the first and the second ones. The number of output data channels became 64 because the depth of conv3d_3 and conv3d_4 was 64.

Note that 25 is not divisible by 2, so the output shape of the max_pooling3d_2 was 12x30x40, which might affect the accuracy of classification in a bad way.

zero_padding3d_5 (ZeroPadding3D)	(None, 14, 32, 42, 64)	0
conv3d_5 (Conv3D)	(None, 12, 30, 40, 128)	221312
batch_normalization_5 (Batch Normalization)	(None, 12, 30, 40, 128)	48
zero_padding3d_6 (ZeroPadding3D)	(None, 14, 32, 42, 128)	0
conv3d_6 (Conv3D)	(None, 12, 30, 40, 128)	442496
batch_normalization_6 (Batch Normalization)	(None, 12, 30, 40, 128)	48
zero_padding3d_7 (ZeroPadding3D)	(None, 14, 32, 42, 128)	0
conv3d_7 (Conv3D)	(None, 12, 30, 40, 128)	442496
batch_normalization_7 (Batch Normalization)	(None, 12, 30, 40, 128)	48
max_pooling3d_3 (MaxPooling3D)	(None, 6, 15, 20, 128)	0

The depth of the rest of the convolutional layers in the network were 128, after the final max pooling layer, the data shape became 6x15x20.

flatten_1 (Flatten)	(None, 230400)	0
dense_1 (Dense)	(None, 512)	117965312
batch_normalization_8 (Batch Normalization)	(None, 512)	2048
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 5)	2565
activation_1 (Activation)	(None, 5)	0
=====		
Total params: 119,273,293.0		
Trainable params: 119,271,897.0		
Non-trainable params: 1,396.0		

This chart consists the details of the last part of my model. The data was flattened before they were applied to the dense layer, so the data shape became $6 * 15 * 20 * 128 = 230,400$. The depth of the dense_1 was 512, and that of the dense_2 was 5, which equals to the number of the classes of action.

5.2.2 Performance

1	Model	Accuracy	Training Loss	Testing loss
2	4-layer CNN (RMSProp, without BN)	0	16.1181	16.1181
3	9-layer CNN (RMSProp, without BN)	0.08	14.5063	14.8286
4	9-layer CNN (SGD, lr 0.1, with BN)	0.45	10.0762	8.865
5	9-layer CNN (SGD, lr 0.0001, with BN)	0.65	1.1853	0.7685

The chart I show above is a comparison of accuracy between different models of mine. In that graph, the 4-layer CNN refers to the first network I mentioned in Chapter 3; it had three convolutional layers and one dense layer. I used RMSProp as the optimiser for that model, and I used categorical cross-entropy as the loss function. The training loss and testing loss kept increasing until they reached 16.1181, and the accuracy remained 0.

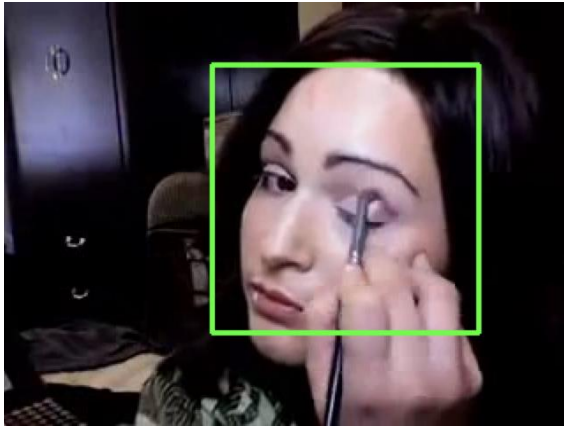
I thought my network was not deep enough to achieve such a difficult task, so I added a few layers to the network, which was the prototype of the second network I described in Chapter 4. After I had trained the model, I realised that it was better than the previous one regarding accuracy, training loss and testing loss. Still, both training loss and testing loss were too high, and the accuracy was too poor to be used as a part of the experiment.

To improve the performance of my model, I did more research on deep learning and other CNN models. Eventually, I found out that the SGD had an extraordinary performance in the deep network, and Batch Normalization can accelerate the training process.^[19] So I put SGD and BN to my network, which became the one I presented in Chapter 4.

Then I train my program with 0.1 learning rate. Thanks to SGD, the accuracy did increase dramatically from 0.08 to 0.45. Both training loss and testing loss decreased reasonably, but they were still too high. I suspected that the learning might be too aggressive, then I tried with 0.0001, and the smaller learning rate gave me another breakthrough. The accuracy went up to 0.65, while both training loss and testing loss remained at 1.0 approximately.

5.3 Object Detection and Object Tracking

5.3.1 Detection



The screenshot on the left is a true positive detection of Haar Cascade. I took face classifier for example, though it is not 100% accurate, it still has a reasonable accuracy.



The screenshots above are the examples of false detection. To capture the ROI correctly, I decided to combine Object Detection and Object Tracking rather than just using one of them. However, this idea cannot cope with the situation that the classifier has a false positive detection (i.e. the screenshot on the right).

Now, it seems that simply using object tracking might be a good idea. Let's have a look at the performance of object tracking.

5.3.2 Tracking



As I explained in Chapter 4, Object Tracking is seeking the maximum pixel density. In my object tracking test, it works perfectly if the size of the window (bounding box) is appropriate. However, it always leads to a failure if the window is too large. The images above are the examples of object tracking.



When the window size is unreasonably large, the algorithm will track something else rather than the object we want. In conclusion, combining object detection and object tracking is not a bad idea, and the result of that can be reliable as long as the classifier gives a true positive detection in the first frame.

5.4 Video Editing

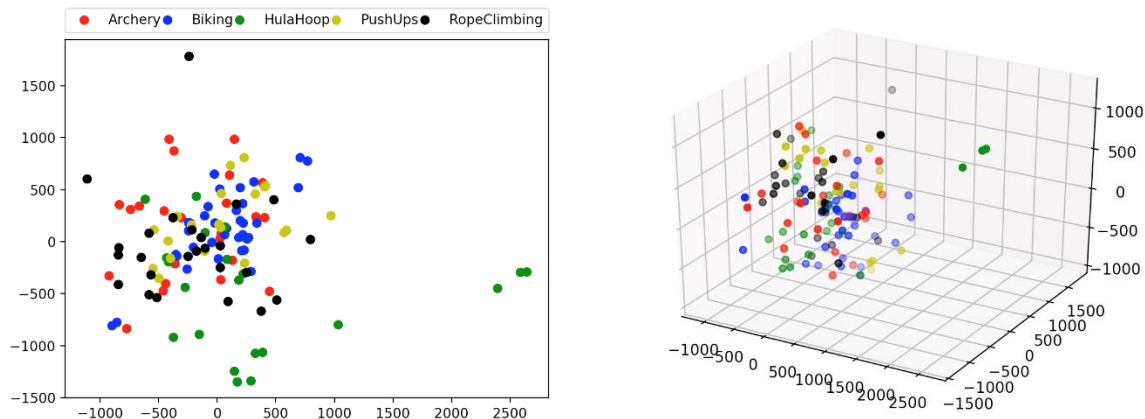


Then I would like to present the modified videos. Using the object detection and object tracking technologies above, I am allowed to manipulate the object in the video since I have the accurate location of that object. There are two different ways to amend the testing videos. I can either setting the pixels to 0, which equals to use

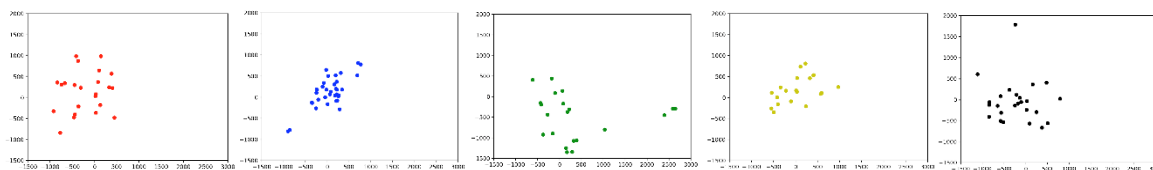
a black box to block the object in the video or replacing the ROI with a different object from other videos.

After modifying the videos, I will apply the videos to the neural network and find out how the network classify actions.

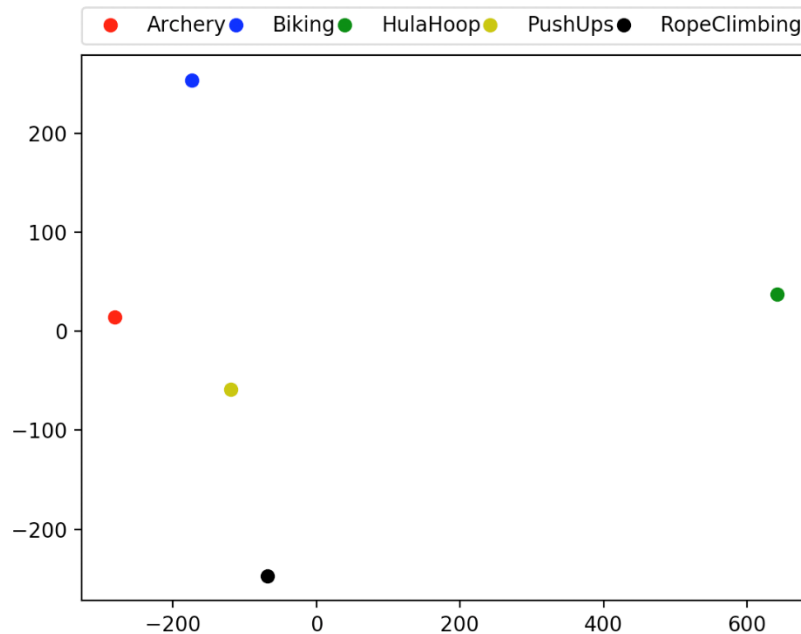
5.5 Visualisation



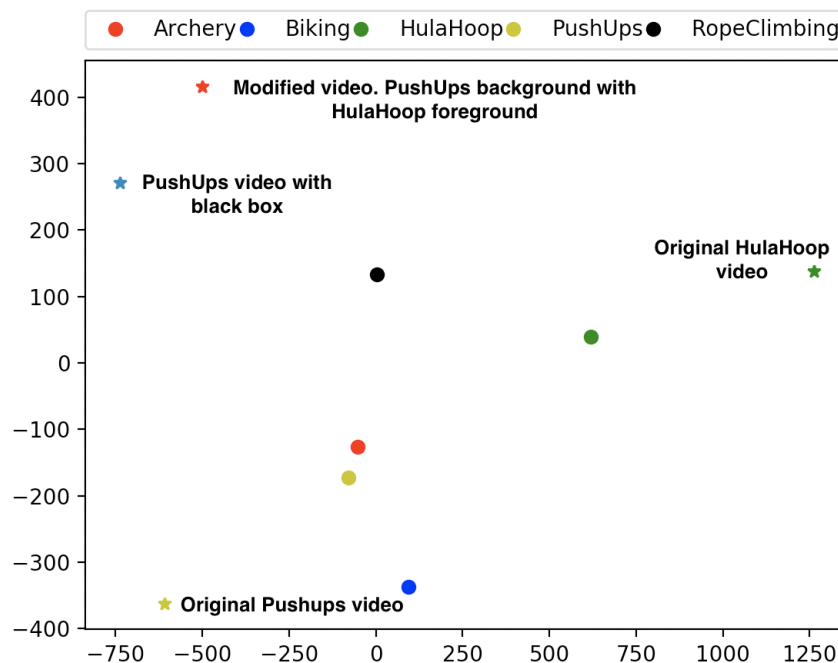
I apply all the testing videos to the network and plot all the data points in both 2D and 3D. Given that they were 5-dimensional data points, which cannot be directly plotted in 2D nor 3D, I used Principal Component Analysis (PCA) to reduce the dimension of the data points. Unfortunately, the regions of different action overlap, and we are not able to draw any conclusion from the results like that.



Then I plot each class of action separately. The good news is that the points belonging to the same class are closed to each other. So I came up with an idea: if I compute the central points of each class in 5-dimensional space first, then apply those central points to PCA, we might be able to separate different class of actions in 2D space.



Fortunately, the idea I described above does separate all the actions. In the graph above, the red point refers to the cluster centre of Archery; the blue point refers to the centre of Biking; the green one refers to Hula Hoop; the yellow one means the centre of Push Ups and the black point is the cluster centre of Rope Climbing. The network will classifier the videos according to the closest cluster centre.



Then I conducted an experiment on my model. I firstly get the prediction result of two testing videos (hula hoop and push up), which are the green star and the yellow star in the graph above. Then I track the object in the push ups testing video and block the ROI by adding a black box (the blue star). Finally, I put a hula hoop action in that black box, which makes it become a video with push up's background and hula hoop's foreground.

From the graph shown above, we notice that the nearest neighbour of the *Original Pushups Video* is action PushUps, the nearest neighbour of the *Original HulaHoop Video* is action HulaHoop. After blocking the ROI of the *Original Pushups Video*, the location of the video is changed. If we add a hula hoop action to that video with a black box, the location will also be changed. To find out more information, I compute the Euclidean Distance from that 4 points to the cluster centres(i.e. Pushups and HulaHoop).

```
Compared to PushUps:
Distance from original action to PushUps: 666.6609203320921
Distance from blocked action to PushUps: 883.4145676080061
Distance from modified action to PushUps: 745.6001402047605
Distance from sample action to PushUps: 1392.5393780562065

Compared to HulaHoop:
Distance from original action to HulaHoop: 1333.820662248894
Distance from blocked action to HulaHoop: 1473.3918760937138
Distance from modified action to HulaHoop: 1229.3896745820177
Distance from sample action to HulaHoop: 788.9702203448821
```

Initially, the distance between the testing video with action push ups and the cluster centre of PushUps is 667. After blocking the ROI of the video, the distance from that video to the cluster centre of push ups becomes 883. Meanwhile, the distance from that video to the cluster centre of hula hoop becomes 1473. By adding a hula hoop action to the video with a black box, the distance to action hula hoop can be decreased, and it is 1229 in this example.

6. Conclusions

I repeat the experiment with different actions. Now I can draw the conclusion from the results of the experiments.

1. It is not guaranteed that object detection/tracking can detect/track the correct object.

Similar to other machine learning model, object detection also has its error rate, which means it cannot guarantee that the right object will be found.

As for object tracking, though it is not using any machine learning algorithm, it would make mistakes if the tracking window is too large. When the window is too large, the background might be considered as the primary object in the window. Then the algorithm might move the window to somewhere else, which is how the mistakes happen.

2. Generally, the required time for training/testing would be increased as the network become deeper.

Generally speaking, the deeper a network is, the longer time would be required for the training process. I designed two different deep convolutional neural networks, though the number of parameters of the deeper number is significantly less than that of the other one, the deeper one still requires much more time on training.

```
8/80 [==>.....] - ETA: 7078s - loss: 2.7076 - acc: 0.2500
16/80 [====>.....] - ETA: 6257s - loss: 2.5900 - acc: 0.3125
24/80 [=====>.....] - ETA: 5465s - loss: 2.4721 - acc: 0.4167
32/80 [=====>.....] - ETA: 4679s - loss: 2.4566 - acc: 0.3750
40/80 [======>.....] - ETA: 3895s - loss: 2.3717 - acc: 0.3750
48/80 [======>.....] - ETA: 3113s - loss: 2.3292 - acc: 0.3542
56/80 [======>.....] - ETA: 2333s - loss: 2.3479 - acc: 0.3393
64/80 [======>.....] - ETA: 1554s - loss: 2.4920 - acc: 0.2969
72/80 [======>...] - ETA: 777s - loss: 2.5432 - acc: 0.3056
80/80 [=====] - 8305s - loss: 2.5648 - acc: 0.3000 - val_loss:
9.8817 - val_acc: 0.1000
```

```
8/80 [==>.....] - ETA: 158674s - loss: 2.0848 - acc: 0.1250
16/80 [====>.....] - ETA: 140918s - loss: 2.1065 - acc: 0.1875
24/80 [=====>.....] - ETA: 123337s - loss: 2.0893 - acc: 0.2500
32/80 [=====>.....] - ETA: 105737s - loss: 2.1733 - acc: 0.2188
40/80 [======>.....] - ETA: 88097s - loss: 2.1773 - acc: 0.1750
48/80 [======>.....] - ETA: 70458s - loss: 2.1473 - acc: 0.1667
56/80 [======>.....] - ETA: 52843s - loss: 2.1292 - acc: 0.1786
64/80 [======>.....] - ETA: 35236s - loss: 2.0459 - acc: 0.2188
72/80 [======>...] - ETA: 17626s - loss: 2.0311 - acc: 0.2083
80/80 [=====] - 176718s - loss: 2.0111 - acc: 0.1875 - val_loss:
1.5242 - val_acc: 0.3000
```

For example, the 9-layer model mentioned in 5.2.1 had 119,273,393 parameters, and the 12-layer model, mentioned in 7.1, had 12,617,725 parameters. However, when they were training with the same data, the 9-layer one required 8,305s for 80 videos, but the 12-layer one required 176,718s. The reason is that the deeper one requires more computation and the number of the parameter does not affect the computation time. In fact, more parameters require more memory to store them instead of more computation time.

3. Learning rate cannot be too aggressive.

In a neural network, the higher learning rate does not mean the model is better than other models with lower learning rate. The learning rate would affect the weights in the network via a function (the functions of SGD and RMSprop are given in 3.3.3), and eventually, the weights would be inappropriate because of high learning rate. However, an extremely small learning rate is still not a good choice. A low learning rate means that the weights are slightly increased or decreased in each iteration. At the beginning of the training process, using such a learning rate would waste computation time.

The best solution is to use a reasonable learning rate and steadily decrease the learning rate. With this idea, The system can update its weights swiftly in the beginning and then search for the best option with a relatively small learning rate.

4. The algorithm of the neural network can be understood as a K-Nearest-Neighbours (KNN) algorithm where K equals 1.

Although the convolutional neural network did not explicitly use KNN algorithm, the softmax activation function would give a nonlinear variant of multinomial logistic regression. In other words, it can also be regarded as converting the values into action probability (in reinforcement learning field) and predicting by the probability. In supervised learning field, it can be regarded as KNN algorithm where $K = 1$.

5. The classification result is affected by both the region of interest (ROI) and the background.

According to the experiments, the classification result will be affected by removing the ROI and replacing the ROI with another action. By computing the Euclidean Distance, I also noticed that the classification result is

combined with the classification result of the background and the classification result of the foreground. Besides, those two results are not equally treated in a classification task. Take push ups for example, the background is more important than the foreground since the distance between the modified video (push up's background and hula hoop's foreground) and push ups class centre is closer than the distance of modified video and hula hoop class centre.

7. Future Work

In the future, I would like to improve my work via three different aspects:

1. Improve my 9-layer VGG-like Convolutional Neural Network by changing the parameters in each layer and the configuration of the network.
2. Try to use Inception-v4 as my network.
3. Use different classifiers for various objects in the videos.

7.1 Improve the Network

After my exploration in Convolutional Neural Network, now I reach a new level of understanding, and I also notice the flaw in my work as well. I would like to describe how I improve my network step by step.

In data pre-processing stage, I would resize all the videos into 47x238x317. Then, for the first layer of the network, I would use 2x5x5 as the parameter in zero-padding, 5x11x11 as the size of the receptive field and 2x3x4 for stride. The shape of output would be 24x80x80, while the output shape of the first layer of my current network is 50x120x160. It is proved that program can dramatically reduce the amount of required memory using such an aggressive convolutional layer in the beginning.

By compromising my network, the required memory used for training process might not exceed the limit of the GPU, which means the training process can be dramatically accelerated (up to 10x faster training with persistent LSTM RNNs^[31]).

Here is the summary of the new Neural Network:

Layer (type)	Output Shape	Param #
zero_padding3d_1 (ZeroPaddin	(None, 51, 248, 328, 3)	0
conv3d_1 (Conv3D)	(None, 24, 80, 80, 32)	58112
batch_normalization_1 (Batch	(None, 24, 80, 80, 32)	96
zero_padding3d_2 (ZeroPaddin	(None, 26, 82, 82, 32)	0
conv3d_2 (Conv3D)	(None, 24, 80, 80, 32)	27680
batch_normalization_2 (Batch	(None, 24, 80, 80, 32)	96
max_pooling3d_1 (MaxPooling3	(None, 12, 40, 40, 32)	0
zero_padding3d_3 (ZeroPaddin	(None, 14, 42, 42, 32)	0

conv3d_3 (Conv3D)	(None, 12, 40, 40, 64)	55360
batch_normalization_3 (Batch Normalization)	(None, 12, 40, 40, 64)	48
zero_padding3d_4 (ZeroPadding3D)	(None, 14, 42, 42, 64)	0
conv3d_4 (Conv3D)	(None, 12, 40, 40, 64)	110656
batch_normalization_4 (Batch Normalization)	(None, 12, 40, 40, 64)	48
zero_padding3d_5 (ZeroPadding3D)	(None, 14, 42, 42, 64)	0
conv3d_5 (Conv3D)	(None, 12, 40, 40, 64)	110656
batch_normalization_5 (Batch Normalization)	(None, 12, 40, 40, 64)	48
max_pooling3d_2 (MaxPooling3D)	(None, 6, 20, 20, 64)	0
zero_padding3d_6 (ZeroPadding3D)	(None, 8, 22, 22, 64)	0
conv3d_6 (Conv3D)	(None, 6, 20, 20, 128)	221312
batch_normalization_6 (Batch Normalization)	(None, 6, 20, 20, 128)	24
zero_padding3d_7 (ZeroPadding3D)	(None, 8, 22, 22, 128)	0
conv3d_7 (Conv3D)	(None, 6, 20, 20, 128)	442496
batch_normalization_7 (Batch Normalization)	(None, 6, 20, 20, 128)	24
zero_padding3d_8 (ZeroPadding3D)	(None, 8, 22, 22, 128)	0
conv3d_8 (Conv3D)	(None, 6, 20, 20, 128)	442496
batch_normalization_8 (Batch Normalization)	(None, 6, 20, 20, 128)	24
max_pooling3d_3 (MaxPooling3D)	(None, 3, 10, 10, 128)	0
flatten_1 (Flatten)	(None, 38400)	0
dense_1 (Dense)	(None, 256)	9830656
dense_2 (Dense)	(None, 1024)	263168
dense_3 (Dense)	(None, 1024)	1049600
dropout_1 (Dropout)	(None, 1024)	0
dense_4 (Dense)	(None, 5)	5125
activation_1 (Activation)	(None, 5)	0
=====		
Total params: 12,617,725.0		
Trainable params: 12,617,521.0		
Non-trainable params: 204.0		

7.2 Inception-ResNet

The Inception architecture has been shown to achieve very good performance at a relatively low computational cost.^[7] The introduction of residual connections in conjunction with a more traditional architecture has yielded state-of-the-art performance in the 2015 ILSVRC challenge, which attracts my attention and arouses my interest to train an Inception-ResNet.

7.3 Different Classifiers for Detection

As I explained in Chapter 6, Haar Cascade is not a good choice for all the objects we need to detect. For example, it usually fails in human body detection. In my test, it not no rare that it cannot successfully identify any human body in a video which has human playing basketball.

In the future, I would like to train another CNN to classify the object in the video. Take VGG-16 for example, my idea is to load the pre-trained weights and train the model with UCF-101 and HMDB-51, then hopefully I can use that model for object detection task in my project.

8. Reflection and Learning

8.1 Reflection

Overall I have enjoyed this project as I have gained a lot of new skills and lessons to take forward into future projects. Inspired by the CUROP project I have done, I become interested in Machine Learning and the similar fields (e.g. Deep Learning). Especially when I was working on the CUROP project, the neural network seemed complicated for me. Thus I did not have any neural network experience until started working on this project.

I am excited that I was able to study neural network as I found it incredibly interesting. By working on this project, I finally have a chance to put what I have learned in the first semester (i.e. Artificial Intelligence and Combinatorial Optimization) into practice and explore more about Deep Learning which is what I want to study.

If I could start the project again, I would do a few things differently. Firstly, I would study on some famous CNNs and explore the theories behind them rather than reading a Deep Learning textbook before case study. Also, since training the model is time-consuming, I would also apply for the privilege to use the supercomputer and take the training course at the beginning of my project. If the computation power is guaranteed, I will construct multiple convolutional neural networks including VGG, Inception Net, ResNet and Inception-ResNet. By comparing the results from different networks, the conclusions would be much more unbiased.

A large amount of time was spent on installing all the tools I need (7 days), pre-processing the data (about 70 hours) and training the networks (more than 200 hours). Although the time spent on installation is inevitable, I could have accelerated the program by using GPU. Without using GPU for acceleration does waste a lot of time. In future projects, if I encounter a similar situation, I would use GPU for computing instead of CPU.

8.2 Learning

This project enabled me to explore the Deep Learning field and use the frameworks such as Keras and TensorFlow. As a student interested in AI, working on this project provides me with a good understanding of the neural network. Also, since it is a complicated project consisting deep neural networks and 3D data,

projects like Natural Language Processing (1D) and Image Classification (2D) seem to be less difficult for me. This project gave me the opportunity to study the state-of-art technologies, though it was confusing and frustrating when I started to learn those cutting-edge technologies, it does give me a chance to improve my researching skills and pace the way for further study in the future.

Time management in the project was vital, to deliver what I had outlined in the initial plan. The main learning point for me while managing the project was how to cope with the unexpected changes to the proposed project timeline. Due to some unforeseen technical problems or personal reasons, the project has to be delayed for a while. For example, when I was implementing video writing function, I failed to save the video file since I did not install OpenCV with ffmpeg support, and OpenCV did not provide any error information, which confused me for a few days. I tried different method to write videos including using Matlab rather than OpenCV, asking the Tech Team to install OpenCV for me in the Labs. The idea of using Matlab did not work because Matlab cannot open .avi files for some reason. I also sent email for installing OpenCV in Linux Lab, but OpenCV in Linux Lab still cannot save the video file because of lacking the ffmpeg support. Eventually, I addressed the problem by myself, but I had already spent too much time on it.

Bibliography

- [1] Y. LeCun (2016). Slide on Deep Learning. Available: <https://indico.cern.ch/event/510372/>
- [2] J. Weng, N. Ahuja and T. S. Huang (1993). Learning recognition and segmentation of 3D objects from 2D images.
- [3] S. Mallick (2017). Object Tracking using OpenCV(C++/Python). Available: <https://www.learnopencv.com/object-tracking-using-opencv-cpp-python/>
- [4] K. Simonyan and A. Zisserman (2014). Very Deep Convolutional Networks for Large-Scale Visual Recognition. Available: http://www.robots.ox.ac.uk/~vgg/research/very_deep/
- [5] A. Krizhevsky, I. Sutskever, G. E. Hinton (2015). ImageNet Classification with Deep Convolutional Neural Networks. Available: http://vision.stanford.edu/teaching/cs231b_spring1415/slides/alexnet_tugce_kyunghee.pdf
- [6] K. He, X. Zhang, S. Ren, J. Sun (2015). Deep Residual Learning for Image Recognition. Available: <https://arxiv.org/abs/1512.03385>
- [7] C. Szegedy, S. Ioffe, V. Vanhoucke, A. Alemi (2016). Inception-v4, Inception-ResNet and the Impact of Residual Connection on Learning. Available: <https://arxiv.org/abs/1602.07261>
- [8] B. Babenko, M. Yang, S. Belongie (2011). Robust Object Tracking with Online Multiple Instance Learning. Available: <http://vision.ucsd.edu/~bbabenko/data/miltrack-pami-final.pdf>
- [9] Tracking with Online Multiple Instance Learning. Available: http://vision.ucsd.edu/~bbabenko/new/project_miltrack.shtml
- [10] Z. Kalal, K. Mikolajczyk, J. Matas (2012). Tracking-Learning-Detection, IEEE Transactions on Pattern Analysis and Machine Intelligence. Available: <http://ieeexplore.ieee.org/document/6104061/>
- [11] A. Rosebrock (2016). macOS: Install OpenCV 3 and Python 3.5. Available: <http://www.pyimagesearch.com/2016/12/05/mac-os-install-opencv-3-and-python-3-5/>
- [12] L. González (2015). Install OpenCV 3 with Python 3 (Mac OSX). Available: <http://luigolas.com/blog/2014/09/15/install-opencv3-with-python-3-mac-osx/>
- [13] J. Boon (2014). Build OpenCV 3 on Mac OSX with Python 3 and ffmpeg support. Available: <http://blog.jiashen.me/2014/12/23/build-opencv-3-on-mac-os-x-with-python-3-and-ffmpeg-support/>

- [14] A. Rosebrock (2016). Ubuntu 16.04: How to install OpenCV. Available: <http://www.pyimagesearch.com/2016/10/24/ubuntu-16-04-how-to-install-opencv/>
- [15] Keras Documentation. Available: <https://keras.io>
- [16] TensorFlow Home Page. Available: <https://www.tensorflow.org>
- [17] Nvidia CUDA Homepage. Available: http://www.nvidia.com/object/cuda_home_new.html
- [18] HDF5 for Python. Available: <http://www.h5py.org>
- [19] S. Ioffe, C. Szegedy (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Available: <https://arxiv.org/pdf/1502.03167.pdf>
- [20] cv::VideoWriter Class Reference. Available: http://docs.opencv.org/master/dd/d9e/class_cv_1_1VideoWriter.html#gsc.tab=0
- [21] A. Rosebrock (2016). Writing to videos with OpenCV. Available: <http://www.pyimage search.com/2016/02/22/writing-to-video-with-opencv/>
- [22] N.Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Available: <http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf>
- [23] I. Sutskever, J. Martens, G. Dahl, G. Hinton (2013). On the importance of initialization and momentum in deep learning. Available: <http://proceedings.mlr.press/v28/sutskever13.pdf>
- [24] J. Duchi, E. Hazan, Y. Singer (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Available: <http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf>
- [25] I. Goodfellow, Y. Bengio, A. Courville (2016). Deep Learning. Available: <http://www.deeplearningbook.org>
- [26] P. Viola, M. Jones (2001). Rapid Object Detection using a Boosted Cascade of Simple Features. Available: http://cgib.nutn.edu.tw:8080/cgit/PaperDL/uang_09032621564947.pdf
- [27] Y. Freund, R. E. Schapire (1996). A decision-theoretic generalization of on-line learning and an application to boosting. Available: <http://cns.bu.edu/~gsc/CN710/FreundSc95.pdf>
- [28] A. Mordvintsev (2013). Meanshift and Camshift. Available: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_video/py_meanshift/py_meanshift.html#meanshift
- [29] G. Bradsky (1998). Real Time Face and Object Tracking as a Component of a Perceptual User Interface. Available: <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=732882>

- [30] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, K. J. Lang (1989). Phoneme Recognition Using Time-Delay Neural Networks. Available: <http://www.cs.toronto.edu/~fritz/absps/waibelTDNN.pdf>
- [31] NVIDIA cuDNN: GPU Accelerated Deep Learning. Available: <https://developer.nvidia.com/cudnn>
- [32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weisds, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau (2011). Scikit-learn: Machine Learning in Python. Available: <http://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf>
- [33] S. Raschka (2014). Implementing a Principal Component Analysis (PCA). Available: http://sebastianraschka.com/Articles/2014_pca_step_by_step.html
- [34] S. Zhou, R. Chellappa, B. Moghaddam (1988). Visual tracking and recognition using appearance-adaptive model in particle filters.
- [35] W. Feller (1971). An introduction to probability theory and its application.
- [36] R. Bracewell (1999). The Fourier Transform and Its Applications.
- [37] M. Marie (2009). Encyclopedia of Distances.
- [38] M. Nielsen (2017). Neural Networks and Deep Learning. Available: <http://neuralnetworksanddeeplearning.com/index.html>
- [39] J. Han, C. Morag (1995). The influence of the sigmoid function parameters on the speed of backpropagation learning.
- [40] P. Verhulst (1845). Mathematical Researches into the Law of Population Growth Increase. Available: http://gdz.sub.uni-goettingen.de/dms/load/img/?PPN=PPN129323640_0018&DMDID=dmdlog7
- [41] V. Nair, G. Hinton (2010). Rectified linear units improve restricted Boltzmann machines. Available: http://machinelearning.wustl.edu/mlpapers/paper_files/icml2010_NairH10.pdf
- [42] Y. LeCun, Y. Bengio, G. Hinton (2015). Deep learning. Available: <http://www.nature.com/nature/journal/v521/n7553/full/nature14539.html>
- [43] X. Glorot, A. Bordes, Y. Bengio (2011). Deep sparse rectifier neural networks. Available: <http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf>
- [44] L. Tóth (2013). Available: <http://www.inf.u-szeged.hu/~tothl/pubs/ICASSP2013.pdf>

- [45] A. Mass, A. Hannun, A. Ng (2014). Rectifier Nonlinearities Improve Nerual Network Acoustic Models. Available: http://web.stanford.edu/~awni/papers/relu_hybrid_icml2013_final.pdf
- [46] M. Christopher (2006). Pattern Recognition and Machine Learning.
- [47] L. Bottou (1998). “Online Algorithms and Stochastic Approximations”. Online Learning and Neural Networks. Available: <http://leon.bottou.org/publications/pdf/online-1998.pdf>
- [48] T. Tieleman, G. Hinton (2012). RMSprop: Divide the gradient by running aveeage of its recent magnitude.
- [49] G. Hinton (2016). Overview of mini-batch gradient descent. Available: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf
- [50] Y. LeCun, L. Bottou, G. Orr, K. Muller (1998). Neural Networks: Tricks of the trade.
- [51] L. Rosasco, E. De Vito, A. Caponnetto, M. Piana, A. Verri (2004). “Are Loss Function All the Same”. Neural Computation. Available: <http://web.mit.edu/lrosasco/www/publications/loss.pdf>
- [52] L. Rosasco, T. Poggio (2014). A regularization Tour of Machine Learning.
- [53] P. Viola, M. Jones (2001). Robust Real-time Object Detection. Available: <http://cite.seerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.110.4868>
- [54] E. Real, J. Shlens, S. Mazzocchi, X. Pan, V. Vanhoucke (2017). YouTube-BoundingBoxes Dataset. Available: <https://research.google.com/youtube-bb/>